Testing interview questions

1.what is PDCA?

PDCA : PDCA stands for plan do check and act.

It is the simple process involved in developing any software application.

Plan : Plan stands for defining the goal and achieving the goal.

Do : Since we have planned the goal, we start doing or executing the requirements in order to achieve the goal.

Check : This is the testing part of the software platform, where we check if we are going according to the plan.

Act : This is a kind of optional phase, if an issue occurs in the check cycle then we act on it to to take the actions accordingly and correct the software.

2.Difference between white box black box and grey box testing?

1. **Whitebox Testing**: Imagine you have a transparent box (like glass) and you can see what's inside. In software, this means the tester knows the internal workings of the code, like the programming logic and structure. So, whitebox testing involves testing with knowledge of the code's internals to check if everything works as expected.
2. **Blackbox Testing**: Imagine you have a closed box and you can't see what's inside. With software, this means the tester doesn't know the internal code details; they're testing the software from the user's perspective. Blackbox testing involves trying different inputs and checking if the outputs are correct, without knowing how the software achieves it.
3. **Graybox Testing**: Think of this as a semi-transparent box. In graybox testing, the tester has some partial knowledge of the internal workings of the code. They might not know everything, but they have some insights. This allows them to design tests that combine both internal and external perspectives.

So, in simple terms:

- Whitebox testing is testing with knowledge of how the code works.
- Blackbox testing is testing without knowing how the code works, just its inputs and outputs.

- Graybox testing is testing with some limited knowledge of how the code works.

3. What are the advantages of designing tests early in the life cycle
    1. **Catching Mistakes Sooner**: Designing tests early helps catch mistakes in the software's foundation. This saves time and effort later on.
    2. **Saving Time and Money**: Fixing problems in software is like fixing a leak in a boat. If you find the leak when it's small, it's easier to patch up. But if you wait, the leak can get bigger and cause more damage. Early tests help identify issues when they're small and less expensive to fix.
    3. **Guiding Development**: Tests act like a map guiding a traveler. When you design tests early, developers know where they're headed. They can build the software with a clear goal in mind, making sure it meets the requirements and functions properly.
    4. **Clear Communication**: Imagine making a recipe. If you have a clear list of ingredients and steps, it's easier to cook. Similarly, having tests designed early provides clear instructions for the developers on what the software should do. It reduces confusion and misunderstandings.
    5. **Building Confidence**: Early tests help ensure that the software works properly right from the start, giving you confidence in its quality.

4. What are the types of defects?
    1. **Functional Defects**: These are issues where the software doesn't perform its intended function correctly. For example, if a calculator app gives wrong results for simple arithmetic operations.
    2. **Performance Defects**: These defects relate to how well the software performs. It could be slow response times, crashes when handling a large number of users, or excessive memory usage.
    3. **Compatibility Defects**: Compatibility issues arise when the software doesn't work well with different operating systems, devices, or browsers. For instance, a website might look broken in certain web browsers.
    4. **Usability Defects**: Usability defects involve problems related to user experience and user interface design. This could include confusing navigation, unclear instructions, or buttons that don't work as expected.

Syntax Defects: Syntax defects are related to the coding structure and syntax errors in the software's source code. These can prevent the code from compiling or executing properly.

Regression Defects: Regression defects occur when a change or update in the software causes unintended issues in previously working areas of the code.

5.what is exploratory testing?

Exploratory testing is like going on an adventure while testing software. Instead of following a fixed path, testers explore the software like an explorer in a new land. They play around, try different things, and see what they discover.

Imagine you're exploring a new video game. You're not just following a guide, but you're trying out different characters, paths, and actions to see what happens. Similarly, in exploratory testing, testers don't just follow a predetermined script; they use their creativity and curiosity to find issues, like bugs or unexpected behaviors.

It's like being a detective in a mystery story. Testers ask questions like "What if I click this button while doing that?" or "What happens if I enter really long text?" By doing this, they can find problems that might not have been thought of before.

Exploratory testing is a bit like testing the unknown, which can help uncover hidden issues that scripted testing might miss. It's a flexible and creative way to make sure the software works well and provides a great experience for users.

6. When should exploratory testing be performed?

Exploratory testing can be performed at various stages :

1. At the early stages of development, exploratory testing can be performed in order to avoid the coding errors,logical errors or glitches. Its like trying to find the problems before they become too big.
2. When a new feature is added to the software we can perform exploratory testing so that we can understand whether the application is working fine or not when new features are added to it.
3. When bugs are fixed, we can perform exploratory testing so that the testers can understand whether the bugs are actually fixed or not,whether they have raised an issue some where else in the software by fixing the current bug.

4. This testing can also be performed at user perspective to check the user interface. It involves checking all the navigations, buttons, inputs etc.

7. Tell me about the risk-based testing.

Risk-based testing is like focusing your attention where the biggest dangers are when crossing a busy street. In software testing, it means concentrating your testing efforts where there's the most potential for problems.

Imagine you're building a sandcastle near the ocean. You know that the closer you build to the water, the higher the chance of your castle getting washed away. So, you decide to build a bit farther back where it's safer. That's similar to how risk-based testing works.

In software, "risks" are things that might go wrong or cause trouble. Risk-based testing involves figuring out which parts of the software are most important, most complex, or have the potential to cause the most trouble if they don't work correctly. You then focus your testing on these risky areas to catch problems early.

For example, if you're testing an online shopping website, you might prioritize testing the payment process because if that doesn't work, the website won't make money. You might also focus on testing security features, as any breach could lead to stolen information.

In simple terms, risk-based testing is about being smart with your testing efforts. Instead of testing everything equally, you put more effort where it matters most to avoid big problems down the road.

8. What is acceptance testing?

Acceptance testing is done to enable a user/customer to determine whether to accept a software product. It also validates whether the software follows a set of agreed acceptance criteria. In this level, the system is tested for the user acceptability.

Types of acceptance testing are:

1. **User acceptance testing**: It is also known as end-user testing. This type of testing is performed after the product is tested by the testers. The user acceptance testing is testing performed concerning the user needs, requirements, and business processes to determine whether the system satisfies the acceptance criteria or not.

2. **Operational acceptance testing**: An operational acceptance testing is performed before the product is released in the market. But, it is performed after the user acceptance testing.
3. **Contract and regulation acceptance testing**: In the case of contract acceptance testing, the system is tested against certain criteria and the criteria are made in a contract. In the case of regulation acceptance testing, the software application is checked whether it meets the government regulations or not.
4. **Alpha and beta testing**: Alpha testing is performed in the development environment before it is released to the customer. Input is taken from the alpha testers, and then the developer fixes the bug to improve the quality of a product. Unlike alpha testing, beta testing is performed in the customer environment. Customer performs the testing and provides the feedback, which is then implemented to improve the quality of a product.

9.what is accessibility testing?

Accessibility testing is like making sure everyone can enjoy a playground. Just like a playground should be safe and fun for everyone, software and websites should be usable by everyone, including people with disabilities.

Imagine a slide that's too steep for some kids or swings that are too high for others to reach. Accessibility testing ensures that software, websites, and apps are designed in a way that people with disabilities can easily use them. This could mean making text readable for people with poor vision, using captions for videos so that people with hearing difficulties can understand, or making buttons big enough for people with limited mobility to tap on.

In short, accessibility testing is like giving everyone a fair chance to use and enjoy digital things, no matter their abilities. Just like we want everyone to enjoy a playground, we want everyone to enjoy technology without any barriers.

10.what is meant by adhoc testing?

Adhoc testing is like trying something on the spot, without a plan or instructions. It's like when you suddenly decide to cook something with the ingredients you have at home, without following a recipe.

In software testing, adhoc testing means testing a piece of software in an unplanned and unstructured way. Testers explore the software without using a

predefined test plan. They might click around, try different actions, and see what happens. It's a bit like an adventure, where you're just exploring to see if you find any unexpected issues or bugs.

Adhoc testing doesn't follow a strict script; it's more about using your intuition and curiosity to find problems that might not have been thought of before. It's like looking for hidden treasure – you don't know exactly where it is, but you're searching everywhere.

11.what is meant by agile testing?

In software development, Agile is a way of building and improving software step by step, in smaller parts. And Agile testing is the process of testing those smaller parts as they're being developed.

Instead of waiting until the end to test everything, Agile testing involves testing small chunks of the software after they're done.

This helps catch problems early, like finding a puzzle piece that doesn't fit before the whole puzzle is finished.

12.what is an API?

An API is like a waiter at a restaurant. Imagine you're at a table (your app) and you want to order something (get data or do something) from the kitchen (another app or system). You don't go into the kitchen yourself – instead, you tell the waiter (API) what you want, and they go to the kitchen, bring your order, and give it to you.

So, an API (Application Programming Interface) is a special way that different software programs can talk to each other. It's like a set of rules that let one program ask another program for information or actions without needing to know all the details of how the other program works. Just like you don't need to know how the kitchen works to get your food!

13.what is meant by automation testing?

Automated testing is like having a robot tester that checks if software works correctly. Instead of having humans do the same tests over and over, you use computer programs to do the testing automatically.

Testers write special programs (scripts) that mimic what a user would do. These scripts can click buttons, enter text, and perform other actions in the software. Then, when the software changes or updates, these scripts can be run automatically to check if everything still works as expected.

14.what is meant by bottom-up testing?

Bottom-up testing is like building a tower from the ground up. Instead of starting with the top and working downwards, you start with the foundation and add each layer on top of the one below it.

In software testing, bottom-up testing means testing the smaller components or modules of a system first and then gradually combining them to test larger parts of the system.

Here's a simple breakdown:

1. **Start Small**: Begin by testing the individual building blocks, like functions, modules, or components. You make sure each piece works well on its own.
2. **Combine Gradually**: As each piece is verified, you start combining them into bigger parts, like integrating modules together. You test how these combined parts work together.
3. **Move Upwards**: This process continues, where you keep combining and testing bigger and bigger chunks of the software until you've tested the entire system.
4. **Complete System Testing**: Finally, you test the entire system as a whole to ensure that everything works together seamlessly.

Bottom-up testing is like building a puzzle by starting with the individual pieces and then gradually forming the complete picture. It's a way to catch issues in smaller components early on and ensure that the whole system functions properly when everything is put together.

15.what is meant by base-line testing?

Baseline testing is like taking a snapshot of your software at a specific point in time. It helps you understand how the software is performing and behaving at that moment, so you can compare it to future versions or changes.

In software, baseline testing is when you test a stable and well-understood version of the software. This becomes a reference point for future tests. If something goes wrong later, you can compare it to the baseline to see what's different.

Baseline testing helps ensure that changes or updates to the software don't introduce unexpected problems. It's a way to keep track of the software's performance over time and make sure it's moving in the right direction.

16.what is benchmark testing?

Benchmark testing is like running a race to see how fast you are compared to others. In software, it's about measuring the performance of your software or system to see how well it stacks up against established standards or competitors.

Imagine you're testing a new car. You take it to a track and drive it as fast as you can. Then, you compare your time to the times of other cars that have raced on the same track. This helps you understand how good your car's performance is compared to the others.

In software, benchmark testing involves running your software under specific conditions and measuring things like speed, efficiency, or resource usage. You then compare these measurements to known standards or similar software to see how well your software performs.

17.which types of testing are important for web testing?

Here are some important types of testing for web applications, explained in simple words:

1. **Functionality Testing**: This is like checking if all the buttons, forms, links, and features on the website work as they should. It's making sure that users can do what they're supposed to do without any problems.
2. **Usability Testing**: Usability testing is about making sure the website is easy to use. It's like ensuring that everything is well-organized, buttons are easy to find, and the navigation is smooth, so users don't get confused.
3. **Compatibility Testing**: Compatibility testing is like making sure your website looks and works fine on different devices and web browsers. It's like ensuring the site is readable and usable whether you're using a computer, a tablet, or a phone, and regardless of whether you're using Chrome, Firefox, or another browser.

4. **Performance Testing**: Performance testing is about checking how fast your website loads and responds. It's like making sure your site doesn't take forever to load and that it can handle many users at the same time without crashing.

18. What is the difference between web application and desktop application in the scenario of testing?

Web Application Testing:

1. **Hosted Online**: Web applications are hosted on servers and users access them through web browsers over the internet.
2. **Cross-Browser Testing**: Web applications need to be tested across different web browsers like Chrome, Firefox, Safari, etc., to ensure they work well on various platforms.
3. **Cross-Device Testing**: As web applications are accessed from various devices like computers, tablets, and phones, testing across different devices and screen sizes is important.
4. **Compatibility Testing**: Ensuring the web application works well on different operating systems (like Windows, macOS, Linux) is crucial.
5. **Network Considerations**: Since web apps rely on internet connections, testing for performance under different network conditions is important.
6. **Security Testing**: Web applications are exposed to potential security risks from the internet, so thorough security testing is necessary.

Desktop Application Testing:

1. **Installed Locally**: Desktop applications are installed directly on a user's computer and run independently of web browsers.
2. **Platform-Specific Testing**: Desktop apps need to be tested on specific operating systems (Windows, macOS, Linux), considering the differences in how they interact with the system.
3. **Offline Capabilities**: Unlike web apps, desktop apps can often work offline. Testing how well the app functions without internet connectivity is important.

4. **UI Consistency**: Since desktop apps have a fixed user interface, testing for consistent behavior across different functionalities is a key focus.
5. **Installation and Uninstallation Testing**: Testing the installation and uninstallation processes is crucial for desktop apps.
6. **Performance Under Local Conditions**: Desktop apps may not have as many network-related concerns, but they should perform efficiently based on the user's computer specifications.

19.difference between verification and validation

Verification	**validation**
1.verification is about confirming that the software is built according to specifications and follows the process correctly	1.validation is about confirming that the final software product meets the actual user needs and requirements.
2.Reviews, inspections, walkthroughs, and static analysis are the steps involved in verification	2.Testing, simulations, user feedback, and dynamic analysis are the steps involved in validation.
3.example : Checking if coding standards are followed, if design meets requirements.	3.example : Checking if the software works as intended, if it meets user expectations.

20. What is the difference between Retesting and Regression Testing?

Regression testing	**Retesting**
1. Regression is a type of software testing that checks the code change does not affect the current features and functions of an application.	1. Retesting is the process of testing that checks the test cases which were failed in the final execution.
2. The main purpose of regression testing is that the changes made to the code should not affect the existing functionalities.	2. Retesting is applied on the defect fixes.
3. Regression testing is also known as generic testing.	3. Retesting is also known as planned testing.

21.what is preventative approach and reactive approach?

Preventative approach	**Reactive appraoch**
1.Identify and eliminate the errors before they occur	1.Detecting and fixing the errors after they are found.
2.Implemented early in the development process	2.Implemented after the defects are found.
3.Prevents costly and time-consuming issues that might arise later.	3.Addresses issues that have been discovered, potentially causing delays and extra effort.

22.what is meant by exit criteria?

Exit criteria in testing are like checkpoints that help determine whether a testing phase is complete and whether the software is ready to move to the next stage of development or release.

Think of it like finishing a painting. Before you hang it on the wall, you might have criteria like ensuring the colors are well-blended, there are no visible mistakes, and the canvas is properly stretched. Similarly, exit criteria in testing are a set of conditions that need to be met to ensure the software is in a good state.

23.why is the decision table testing is used?

Decision table testing is used to systematically test multiple combinations of inputs and their corresponding outputs based on various conditions and rules. It's particularly effective when dealing with complex business logic, rules, or scenarios where different inputs lead to different outcomes. Decision table testing helps ensure that all possible combinations are tested, making it easier to catch defects and validate the behaviour of the system.

1. **Complex Scenarios**: When you have many inputs and conditions that affect the outcome, decision table testing helps ensure you cover all possible combinations without missing any.
2. **Coverage**: Decision tables help achieve thorough test coverage, as they guide testers to consider all permutations of inputs and conditions.
3. **Clarity**: Decision tables provide a visual representation of rules and their interactions, making it easier to understand the logic being tested.

24.what is alpha and beta testing?

No.	Alpha Testing	Beta Testing
1)	It is always done by developers at the software development site.	It is always performed by customers at their site.
2)	It is also performed by Independent testing team	It is not be performed by Independent testing team
3)	It is not open to the market and public.	It is open to the market and public.
4)	It is always performed in a virtual environment.	It is always performed in a real-time environment.
5)	It is used for software applications and projects.	It is used for software products.
6)	It follows the category of both white box testing and Black Box Testing.	It is only the kind of Black Box Testing.
7)	It is not known by any other name.	It is also known as field testing.

25.what is meant by monkey testing?

Monkey testing is like letting a playful monkey loose in a room to see what it interacts with. In software, it means allowing a program or system to be tested randomly, without following any specific plan or script. Just like a curious monkey might touch, play, and explore different things, monkey testing aims to uncover unexpected issues by randomly interacting with the software.

Imagine you have a new video game. Instead of following a guide or playing in a specific way, you randomly press buttons, move the joystick, and see what happens. You might discover some hidden bugs or glitches that scripted testing might not find.

In software, monkey testing involves inputting random data, clicking buttons in a random order, and performing various actions without a predetermined plan. This helps identify vulnerabilities, crashes, or unexpected behaviours that might occur when users interact with the software in unpredictable ways.

26. What is the negative and positive testing?

Negative Testing: When you put an invalid input and receive errors is known as negative testing.

Positive Testing: When you put in the valid input and expect some actions that are completed according to the specification is known as positive testing.

27.what is meant by boundary value analysis testing?

Boundary value analysis (BVA) is a software testing technique used to identify and evaluate the behavior of a software application at the boundary conditions of its input domain. The input domain refers to the range of valid input values that a software component or system can accept.

The primary goal of boundary value analysis is to discover defects or vulnerabilities that may occur at or near the edges of this input domain. These defects are often more likely to cause problems

1. Identify Input Boundaries: First, you need to identify the input boundaries for the specific parameter or variable you are testing. These boundaries include the minimum and maximum values, as well as the values immediately before and after these boundaries.
2. Test at Boundaries: Design test cases that use values exactly on these boundaries. For example, if a parameter should accept values from 1 to 10, you would test with values like 1, 10, and any other values that are right at the edge, such as 0 and 11.
3. Test Just Inside the Boundaries: Create additional test cases with values just inside the boundaries. For example, if the boundary values are 1 and 10, test with values like 2 and 9 to ensure that the software behaves correctly when input values are close to the edges.
4. Test Just Outside the Boundaries: Also, create test cases with values just outside the boundaries. For instance, if the boundaries are 1 and 10, test with values like 0 and 11 to see how the software handles inputs that are slightly out of range.

28. How would you test the login feature of a web application?

There are many ways to test the login feature of a web application:

- Sign in with valid login, Close browser and reopen and see whether you are still logged in or not.

- Sign in, then log out and then go back to the login page to see if you are truly logged out.
- Log in, then go back to the same page, do you see the login screen again?
- Session management is important. You must focus on how do we keep track of logged in users, is it via cookies or web sessions?
- Sign in from one browser, open another browser to see if you need to sign in again?
- Log in, change the password, and then log out, then see if you can log in again with the old password.

29. What are the types of performance testing?

Performance testing: Performance testing is a testing technique which determines the performance of the system such as speed, scalability, and stability under various load conditions. The product undergoes the performance testing before it gets live in the market.

1. Load testing:

- Load testing is a testing technique in which system is tested with an increasing load until it reaches the threshold value.

Note: An increasing load means the increasing the number of users.

- The main purpose of load testing is to check the response time of the system with an increasing amount of load.
- Load testing is non-functional testing means that the only non-functional requirements are tested.
- Load testing is performed to make sure that the system can withstand a heavy load

2. Stress testing:

- Stress testing is a testing technique to check the system when hardware resources are not enough such as CPU, memory, disk space, etc.
- In case of stress testing, software is tested when the system is loaded with the number of processes and the hardware resources are less.
- The main purpose of stress testing is to check the failure of the system and to determine how to recover from this failure is known as recoverability.

- Stress testing is non-functional testing means that the only non-functional requirements are tested.

3. Spike testing:

- Spike testing is a subset of load testing. This type of testing checks the instability of the application when the load is varied.
- There are different cases to be considered during testing:
    - The first case is not to allow the number of users so that the system will not suffer heavy load.
    - The second case is to provide warnings to the extra joiners, and this would slow down the response time.

4. Endurance testing:

- Endurance testing is a subset of load testing. This type of testing checks the behavior of the system.
- Endurance testing is non-functional testing means that the only non-functional requirements are tested.
- Endurance testing is also known as Soak testing.
- Endurance testing checks the issues such as memory leak. A memory leak occurs when the program does not release its allocated memory after its use. Sometimes the application does not release its memory even after its use and this unusable memory cause memory leak. This causes an issue when the application runs for a long duration.
- Some of the main issues that are viewed during this testing are:
    - Memory leaks occurred due to an application.
    - Memory leaks occurred due to a database connection.
    - Memory leaks occurred due to a third party software.

5. Volume testing:

- Volume testing is a testing technique in which the system is tested when the volume of data is increased.
- Volume testing is also known as flood testing.
- Volume testing is non-functional testing means that the only non-functional requirements are tested.

- For example: If we want to apply the volume testing then we need to expand the database size, i.e., adding more data into the database table and then perform the test.

6. Scalability testing

- Scalability testing is a testing technique that ensures that the system works well in proportion to the growing demands of the end users.
- Following are the attributes checked during this testing:
  - Response time
  - Throughput
  - Number of users required for performance test
  - Threshold load
  - CPU usage
  - Memory usage
  - Network usage

30.what is the difference between functional and non functional testing?

Functional testing	Non functional testing
1.Functional testing is a testing technique that checks the functionality of the application.	1.Non functional testing checks non functional aspects such as stability,efficiency etc.
2.It is implemented before non functional testing	2.It is implemented after functional testing.
3.It mainly focus on customer requirements.	3.It focus on customer expectations.
4.Types : • Unit testing • Integration testing • Acceptance testing	4.Types : • Stress testing • Volume testing • Load testing
5.It can be performed by manual testing.	5.It cannot be performed by manual testing.

31.what is the difference between static and dynamic testing

Static testing	Dynamic testing
1.It is white box testing which is performed at the initial stage of SDLC.	1.Dynamic testing is a testing technique which is done at later stage in SDLC.
2.It is performed before code deployment.	2.It is performed after code deployment.
3.It is implemented at verification stage.	3.It is implemented at validation stage.
4.Execution of code is not done here.	4.Execution of code is compulsory.
5.Checklist is made for testing process.	5.Test cases are executed.

32.what is the difference between positive and negative testing

Positive testing	Negative testing
1.It is the process of testing the application by providing valid data.	1.It is the process of testing the application by providing invalid data.
2. In case of positive testing, tester always checks the application for a valid set of values.	2. In the case of negative testing, tester always checks the application for the invalid set of values.
3. The positive testing tries to prove that the project meets all the customer requirements.	3. The negative testing tries to prove that the project does not meet all the customer requirements.

33. What are the different models available in SDLC?

There are various models available in software testing, which are the following:

- Waterfall model
- Spiral Model
- Prototype model
- Verification and validation model

34.what is smoke testing

Smoke testing, in simple terms, is like a quick checkup for software to see if it's healthy and ready for more comprehensive testing. It's called "smoke testing" because it's similar to checking if there's any smoke when you start a machine – if there's smoke, it's a sign that something might be seriously wrong.

1. **Basic Testing**: During smoke testing, testers or developers perform a set of basic tests on the software. These tests are typically simple and cover the most fundamental functions of the software.
2. **Quick Evaluation**: The goal is not to dive deep into every feature but to quickly evaluate if the software can perform its core functions without major errors. If there are critical issues at this stage, it's a sign that the software might not be stable enough for more detailed testing.
3. **Go/No-Go Decision**: Based on the results of the smoke test, a decision is made. If the software passes the smoke test (meaning no serious issues or "smoke" is detected), it's considered stable enough to proceed with more thorough testing. If there are significant problems (the "smoke alarm" goes off), the software might be sent back to development for fixes before further testing.

35.what is sanity testing

Sanity testing, in simple terms, is like a common-sense check for software. It's a quick test to see if the most important parts of the software are working reasonably well before diving into more detailed testing.

Here's how it works:

1. **Focus on Key Features**: During sanity testing, testers or developers concentrate on the most critical features of the software. These are the functions that must work properly for the software to be considered usable.
2. **Basic Evaluation**: Testers perform a limited set of tests on these key features to ensure they are not seriously broken. It's not an in-depth examination; it's more like a quick glance to check if things seem okay.
3. **Decision to Proceed**: Based on the results of sanity testing, a decision is made. If the critical features are working reasonably well, it's a sign that the software is in good enough shape to proceed with more comprehensive testing. If there are severe issues with these crucial functions, further testing may be postponed until these issues are fixed.

36.what is dry run testing?

A dry run test, in simple words, is like practicing or simulating a process without actually doing it for real. It's a way to check if a plan or a series of steps will work correctly before you commit to doing it for real.

Here's how it works:

1. **Simulated Execution**: In a dry run test, you go through the steps of a process or a task without actually carrying it out. You pretend to do it, but you don't perform the real actions or make real changes.
2. **Check for Errors**: During this practice run, you carefully review each step to check for mistakes, problems, or unexpected issues. It's like proofreading or double-checking your plan.
3. **Identify Corrections**: If you find any errors or problems during the dry run, you can make corrections and adjustments to the plan before actually executing it for real. This helps prevent mistakes and ensures that the process will go smoothly.
4. **Boost Confidence**: Dry runs help build confidence. When you practice something beforehand, you feel more prepared and less likely to encounter surprises when you do it for real.

Common terms used in testing

1. **Test Case**: A test case is like a detailed recipe for testing a specific aspect of software. It includes step-by-step instructions on what to do, what inputs to use, and what results to expect.
2. **Test Suite or Test Scenario**: Think of a test suite as a collection of test cases grouped together. It helps organize and execute multiple tests efficiently.
3. **Test Plan**: A test plan is like a project blueprint. It outlines the overall strategy for testing, including what to test, how to test, and when to test.
4. **Regression Testing**: This is like checking if changes in the software (updates or fixes) haven't broken existing features. It ensures that old stuff still works after new changes are made.
5. **Black Box Testing**: Black box testers don't need to know how the software works internally. They focus on testing its inputs and outputs, like a user would.

6. **White Box Testing**: White box testers, on the other hand, examine the internal workings of the software. They check the code and logic to find issues.
7. **Functional Testing**: This type of testing is like evaluating whether the software does what it's supposed to do. It checks if it meets its functional requirements.
8. **Non-Functional Testing**: This is about evaluating how well the software performs, like its speed, security, and usability. It's not just about what it does but how it does it.
9. **Smoke Testing**: As mentioned earlier, smoke testing is a quick check to see if the software starts without major issues, ensuring it's "smoke-free" and ready for more testing.
10. **Sanity Testing**: This is a brief check to see if the core functions of the software are working reasonably well, ensuring it's "sane" before diving deeper into testing.
11. **Bug**: A bug is an error or problem in the software. It's something that doesn't work as intended.
12. **Test Environment or Testbed**: This is like the setting in which you perform tests. It includes the hardware, software, and data needed for testing.
13. **Test Data**: Test data is like the input values you use in your tests. It can be both normal data and unusual or edge-case data.
14. **Test Execution**: This is when you actually run the tests you've planned. You follow the test cases and observe the results.
15. **Defect Report**: When testers find bugs, they create defect reports. These reports detail what the bug is, how to reproduce it, and its severity.
16. **Test Automation**: Test automation is like using robots to run tests. Instead of doing tests manually, you write scripts to automate the testing process.
17. **Test Coverage**: This measures how much of the software's code is tested. Think of it as checking which parts of a maze you've explored.
18. **User Acceptance Testing (UAT)**: This is like giving the software to actual users to test and see if it meets their needs and expectations.
19. **Load Testing**: Load testing is like checking how a bridge handles heavy traffic. It tests how the software performs under a heavy workload.

20. **Alpha and Beta Testing**: Alpha testing is like checking a new recipe in your kitchen. Beta testing is like letting friends taste it before serving it to a bigger audience. Alpha is testing within the organization, while beta is testing with a small group of external users.

What is a bug?

A bug in testing is like a mistake or issue found in the software being tested. Testers perform various tests to check if the software works correctly and meets its requirements. When they encounter something that doesn't work as expected, they report it as a bug. These bugs can include things like:

1. **Functional Bugs**: When the software doesn't perform a specific function as described in its requirements or specifications.
2. **Crashes**: If the software unexpectedly stops working or crashes during testing, it's considered a bug.
3. **Incorrect Output**: When the software produces the wrong results or outputs different information than it should.
4. **Performance Problems**: Bugs related to how fast or efficiently the software runs, such as slow response times or excessive resource usage.
5. **User Interface Issues**: Problems with how the software's interface looks or behaves, such as buttons not working or visual glitches.

We have various bug tracking tools available in the market, such as:

- Jira
- Bugzilla
- Mantis
- Telelogic

Why does an application have bugs?

1. **Complexity**: Software can be incredibly complex, with thousands or even millions of lines of code. Managing and coordinating all this code can lead to errors.

2. **Human Error**: Software development involves people, and people make mistakes. Developers may overlook or misunderstand requirements, leading to coding errors.
3. **Time Constraints**: Pressure to release software quickly can lead to shortcuts or rushed coding, increasing the likelihood of errors.
4. **Changing Requirements**: As project requirements evolve or new features are added, it can be challenging to ensure that existing code still works correctly.
5. **Integration Issues**: When different components or libraries are integrated into an application, they may not always work seamlessly together, causing unexpected bugs.
6. **Hardware and Software Variability**: Applications must run on various devices, operating systems, and configurations. These differences can lead to compatibility issues and bugs.
7. **Testing Gaps**: Incomplete or inadequate testing can fail to catch bugs. Exhaustive testing is often impractical due to time and resource constraints.

What is the difference between testcase and usecase?

Test Case	Use case
1.Defines specific steps and conditions for testing a particular aspect or feature of the software.	1.Describes how the system interacts with users or external systems to achieve a specific goal or function.
2.Focused on testing and validation.	2.Focused on capturing system behavior and functionality.
3.Used by testers to verify that the software functions correctly.	3.Used by developers, designers, and business analysts to understand and design system functionality.
4.Focuses on validation and verification.	4.Focuses on capturing functional requirements and user interactions.

Manual testing

Manual testing is a software testing approach in which a human tester performs tests on a software application without the use of automation tools or scripts. Instead of relying on automated test scripts, manual testers execute test cases by hand, interact with the application's user interface, and observe its behavior to identify defects or issues.

Difference between Manual testing and Automation Testing

Aspect	Manual Testing	Automation Testing
Human Involvement	Testing is performed manually by human testers.	Testing is performed using automated test scripts and tools.
Speed and Efficiency	Slower and less efficient for repetitive and extensive testing.	Faster and more efficient for repetitive and regression testing.
Initial Setup Time	Requires less initial setup time since test cases are executed manually.	Requires more initial setup time to create and maintain test scripts.
Test Coverage	May have limitations in achieving comprehensive test coverage.	Can provide comprehensive test coverage, especially for repetitive tasks.
Reusability	Test cases cannot be easily reused without manual execution.	Test scripts can be reused across multiple test cycles and environments.
Execution Flexibility	Adaptable to exploratory, usability, and ad-hoc testing.	Best suited for repetitive, regression, and performance testing.
Human Error	Prone to human error and subjectivity in test execution and reporting.	Less susceptible to human error in test execution and reporting.

Important

What are the different types of exceptions present in selenium webdriver

1. **NoSuchElementException**: This exception is thrown when the WebDriver cannot locate a web element using the specified locator strategy (e.g., **findElement(By.xxx)**). It typically occurs when the element is not present on the web page or the locator used is incorrect.
2. **TimeoutException**: A TimeoutException is raised when a specific operation (e.g., waiting for an element to appear) times out before the expected condition is met. It can be caused by slow page loading or the element taking longer to appear than expected.
3. **ElementNotVisibleException**: This exception occurs when WebDriver attempts to interact with an element that is present in the DOM but not visible on the web page. For example, trying to click on a hidden element.

4. **ElementNotSelectableException**: This exception is thrown when WebDriver tries to interact with an element that is not selectable, such as trying to select an option in a dropdown when it is disabled.
5. **StaleElementReferenceException**: This exception is raised when a web element is no longer attached to the DOM, usually because the page has been refreshed or the element has been removed or replaced.
6. **InvalidElementStateException**: This exception occurs when an action is performed on a web element that is in an invalid state. For example, trying to input text into a read-only input field.
7. **WebDriverException**: WebDriverException is a general exception class for WebDriver-related issues that don't fall into more specific exception categories.

What is page object model?

The Page Object Model (POM) is a design pattern used in software automation testing, particularly in web application testing using tools like Selenium WebDriver. It is designed to enhance test code maintainability, readability, and reusability by encapsulating the interactions with web pages as individual objects.

Each web page or a section of a web page is represented as a separate class called a "Page Object." These classes contain methods and properties that define the elements and actions available on that page. Page Objects act as a bridge between the test code and the web elements on the page.

Page Object classes contain methods that define interactions with the web elements on the page. For example, if you have a login page, the Page Object for that page would include methods like **enterUsername**, **enterPassword**, **clickLoginButton**, etc., which encapsulate the actions you can perform on the login page.

which type of data source can be used for selenium framework?

In a Selenium test automation framework, you may need to use various types of data sources to provide input data for your tests, parameterize test cases, and drive data-driven testing. Here are some common types of data sources that can be used in a Selenium framework:

1. **CSV (Comma-Separated Values) files:** CSV files are plain text files where data is separated by commas or other delimiters. You can read data from CSV files and use it to supply test data or test case parameters.
2. **Excel spreadsheets:** Excel files, such as XLSX or XLS files, are widely used for storing structured data. Selenium can interact with Excel files using libraries like Apache POI (for Java), openpyxl (for Python), or other Excel libraries in different programming languages.
3. **JSON (JavaScript Object Notation) files:** JSON is a lightweight data-interchange format. You can use JSON files to store and organize test data or configuration settings. Most programming languages provide built-in support for parsing JSON.
4. **XML (Extensible Markup Language) files:** XML is a structured data format that can be used for configuration files or storing test data. You can parse XML files to extract data for your tests.
5. **Databases:** Selenium tests can interact with databases to retrieve or update data. You can use SQL queries or database APIs (e.g., JDBC for Java) to connect to databases and retrieve data dynamically during test execution.
6. **Text files:** Plain text files can be used to store and read data, especially when the data is relatively simple and doesn't require a more structured format like CSV.
7. **Properties files:** Properties files (**.properties** or **.ini** files) are often used to store configuration settings for a Selenium framework. They are simple key-value pairs that can be easily read by most programming languages.

What are the listeners in selenium?

Listeners are like "observers" that watch over your automated tests and take special actions when certain events happen during the tests. Imagine them as "watchdogs" for your tests.

For example, when you run a test, events like "test started," "test passed," "test failed," or "test skipped" occur. Listeners can be set up to respond to these events and do things like:

1. **Logging:** Record what's happening during the test, like when it starts or if it encounters an error.

2. **Screenshots:** Take pictures of the screen when a test fails so you can see what went wrong.
3. **Custom Actions:** Perform specific actions before or after a test, like setting up test data or cleaning up after the test is done.
4. **Reporting:** Create detailed reports about the test results, making it easy to understand what happened.
5. **Integration:** Connect your tests with other tools or services, like sending notifications or storing test results in a database.

TestNG Listeners (Java): TestNG is a popular testing framework for Java, and it provides a built-in mechanism for implementing listeners. Commonly used TestNG listeners include:

- **ITestListener:** This listener captures events related to test methods, such as test start, test success, test failure, and test skipped.
- **ISuiteListener:** It captures events related to test suites, such as suite start and suite finish.
- **InvokedMethodListener:** This listener captures events related to individual test methods, including method start and method finish.

is it possible to go forward and backward in a webpage using selenium webdriver explain?

Yes, it's possible to navigate forward and backward in a webpage using Selenium WebDriver, just like you can with a web browser's navigation buttons (forward and backward arrows). Here's a simple explanation of how to do it:

1. **Navigating Backward (Previous Page):**
    - In Selenium WebDriver, you can use the **navigate().back()** method to go back to the previous page.
    - It simulates clicking the browser's "Back" button and takes you to the page you were on before.

Example:

```
WebDriver driver = new ChromeDriver();

driver.get("https://example.com");
```

// Perform actions on the first page

driver.navigate().back(); // Go back to the previous page

Navigating Forward (Next Page):

- You can use the **navigate().forward()** method to go forward to the next page if you've previously gone back.
- This simulates clicking the browser's "Forward" button.

Example :

WebDriver driver = new ChromeDriver();

driver.get("https://example.com");

driver.navigate().back(); // Go back to the previous page

// Perform actions on the previous page

driver.navigate().forward();

How do you take a screenshot in selenium webdriver?

- We set the path to the ChromeDriver executable (replace **"path_to_chromedriver.exe"** with the actual path to your ChromeDriver executable).
- We initialize the WebDriver for the Chrome browser.
- We navigate to a webpage (replace **"https://example.com"** with the URL of the page you want to capture).
- We use the **getScreenshotAs(OutputType.FILE)** method to capture the screenshot as a File object.
- We use the Apache Commons IO library (**FileUtils.copyFile**) to save the screenshot to a file with a specified name and path.

What types of testing can be achieved using selenium web driver?

1. **Functional Testing:** Selenium is most commonly used for functional testing, which involves verifying that the application's features and functions work as expected. This includes testing individual features, user interactions, and user flows.

2. **Regression Testing:** Selenium is well-suited for regression testing, where you ensure that new code changes or updates do not introduce new defects or break existing functionality. Automated regression tests help maintain software quality during continuous development.
3. **Integration Testing:** Selenium can be used to perform integration testing, where you validate that different components or modules of your application work together seamlessly. It's particularly useful for web applications that rely on external services or APIs.
4. **User Interface (UI) Testing:** Selenium allows you to test the user interface of your web application, verifying that the layout, design, and user interactions match the expected behavior. This includes testing buttons, forms, links, and more.
5. **Cross-Browser Testing:** Selenium is often used to perform cross-browser testing, ensuring that your web application works correctly across different web browsers such as Chrome, Firefox, Edge, and Safari. This is crucial for delivering a consistent user experience.

What is meant by assertion in selenium?

In Selenium, an assertion is like a checkpoint or a test that you include in your automated test script to verify if something on a web page is behaving as expected. It's a way to confirm that your web application is working correctly during the test.

What are the different build phases involved in maven?

In Apache Maven, the build process is divided into different phases, and each phase represents a specific step in the software build lifecycle. These phases ensure that a project is built, tested, and packaged in a structured and consistent manner. Here are the primary build phases involved in Maven:

1. **Validate:** In this phase, Maven validates the project's structure, dependencies, and other configuration details. It checks if the project is set up correctly and can proceed with the build process.
2. **Compile:** During this phase, Maven compiles the source code of your project, typically located in the **src/main/java** directory. It compiles the code into bytecode, which is executed by the Java Virtual Machine (JVM).

3. **Test:** In this phase, Maven runs the unit tests associated with your project. Test classes are typically located in the **src/test/java** directory. Maven uses testing frameworks like JUnit or TestNG to execute these tests.
4. **Package:** The package phase creates a distributable package or artifact of your project. The type of artifact generated (e.g., JAR, WAR, or ZIP) depends on the project's packaging configuration.
5. **Verify:** This phase performs additional checks and verifications on the project, such as integration tests or code quality checks. It ensures that the project meets specified quality standards.
6. **Install:** During the install phase, Maven copies the project's artifact (e.g., JAR or WAR file) to a local repository on your development machine. This local repository is used to share dependencies and artifacts across different projects on your local system.
7. **Deploy:** The deploy phase involves copying the project's artifact to a remote repository, typically a central repository like Maven Central or a corporate repository. This phase is used for sharing the project's artifact with other developers or teams.

how do you handle keyboard and mouse actions using selenium?

Selenium WebDriver provides the Actions class to handle keyboard and mouse actions, allowing you to perform various interactions with web elements. Here are some common keyboard and mouse actions you can perform using Selenium WebDriver:

Clicking: Use the **click()** method to simulate a left-click action on a web element, such as a button or a link.

```
WebElement element = driver.findElement(By.id("elementId"));

Actions actions = new Actions(driver);

actions.click(element).build().perform();
```

Double-Clicking: Use the **doubleClick()** method to simulate a double-click action on a web element.

```
WebElement element = driver.findElement(By.id("elementId"));

Actions actions = new Actions(driver);

actions.doubleClick(element).build().perform();
```

Right-Clicking: Use the **contextClick()** method to simulate a right-click (context menu) action on a web element.

```
WebElement element = driver.findElement(By.id("elementId"));

Actions actions = new Actions(driver);

actions.contextClick(element).build().perform();
```

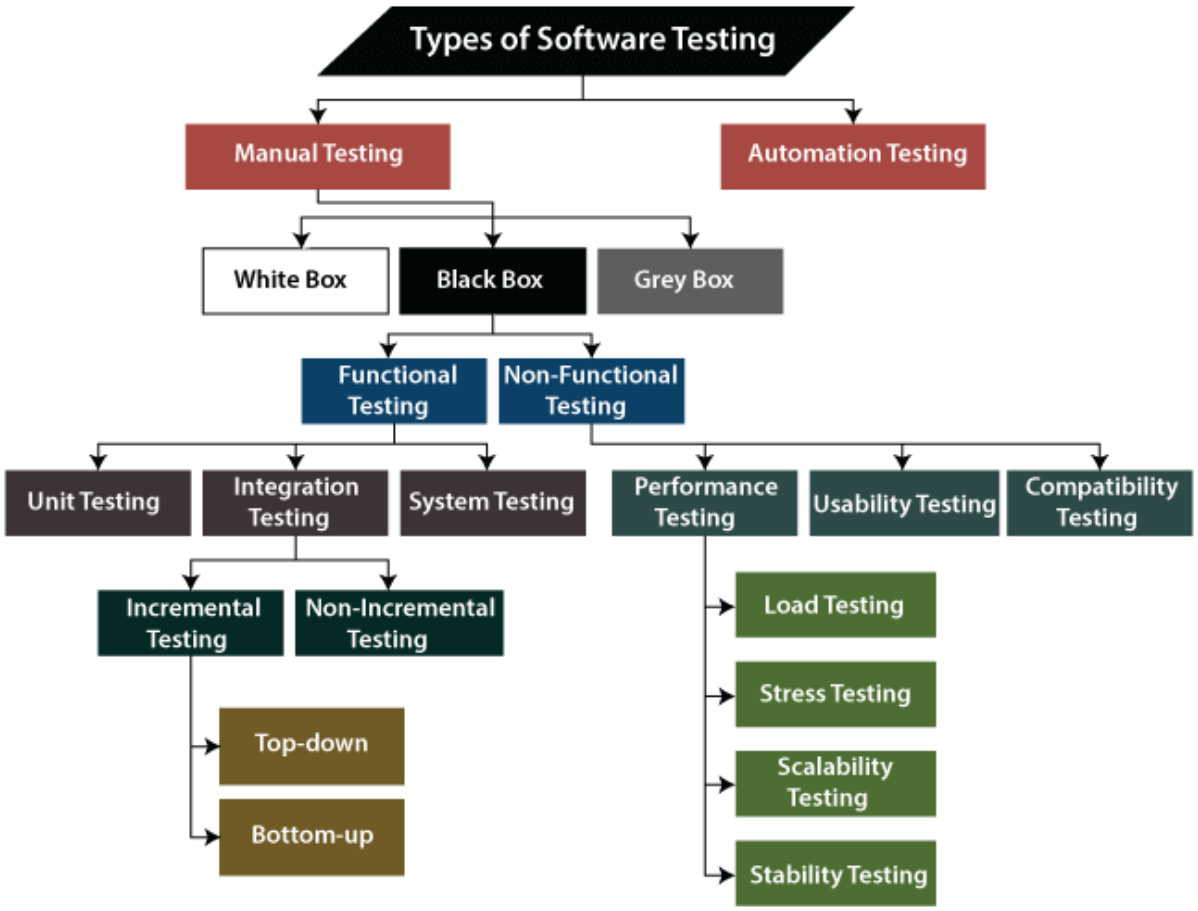
Hovering (Mouse Over): Use the **moveToElement()** method to simulate hovering over a web element, which can trigger dropdown menus or tooltips.

```
WebElement element = driver.findElement(By.id("elementId"));

Actions actions = new Actions(driver);

actions.moveToElement(element).build().perform();
```

what is the difference between http and https

Feature	HTTP	HTTPS
Security	Not secure (plain text)	Secure (encrypted)
Data Encryption	No encryption	Data encrypted
Data Integrity	Not guaranteed	Data integrity
URL Prefix	http://	https://
Port	Default port 80	Default port 443
Usage	Used for regular websites and applications	Used for secure transactions, online banking, e-commerce, etc.
SSL/TLS Required	No	Yes
Certificate	Not required (optional)	Required (SSL/TLS certificate)
Cost	Generally cheaper or free	Cost associated with obtaining and renewing SSL/TLS certificates
Browser Indication	Not indicated	Indicates a secure connection with a padlock icon and "https://"

Regen



What do you mean by pesticide paradox?

The pesticide paradox is a simple idea in software testing that suggests that if you keep using the same set of test cases over and over again, they will become less effective at finding new bugs in the software. It's similar to how pests can become resistant to pesticides if the same pesticide is used repeatedly.

Here's a simple explanation:

Imagine you have a garden, and you use a specific pesticide to get rid of bugs. At first, it works really well, and you see fewer bugs in your garden. However, over time, some bugs might survive because they are naturally resistant to that pesticide. These surviving bugs can reproduce, and their offspring may also be resistant to the pesticide. So, if you keep using the same pesticide all the time, it becomes less effective at controlling the bug population because the bugs have adapted to it.

In software testing, it's like saying that if you always use the same tests to check a program, the program might become better at handling those tests, but it might still have new or different bugs that those tests don't catch. To prevent this, testers

need to regularly update and create new test cases to find and identify new types of bugs in the software, just like how gardeners might switch to different pesticides to control resistant pests. This way, the testing process remains effective and continues to find issues in the software.

Methods for avoiding the pesticide conundrum include:

To create a completely new set of test cases to put various aspects of the software to the test.

To create new test cases and incorporate them into existing test cases.

What is meant by API Testing?

API testing, in simple terms, is like checking if different parts of a computer program or system are talking to each other correctly.

Imagine you have a bunch of machines (software components) in a factory (your application), and they need to communicate to build a product (perform a task). They talk to each other through a set of buttons and levers (APIs). API testing is like checking if these buttons and levers work as they should.

For example, if one machine (component) needs to send a particular type of material (data) to another machine, you want to make sure it's sending the right stuff in the right way (correct data and format). API testing helps ensure that the communication between these machines (components) is smooth and efficient, just like checking if all the buttons and levers in your factory are doing their job properly.

There are several tools available for API testing, and the choice of tool often depends on your specific requirements, programming language, and preferences. Here are some popular API testing tools:

1. **Postman:** Postman is a widely used and user-friendly tool for testing APIs. It provides a graphical user interface (GUI) that allows you to create and send API requests, view responses, and automate testing workflows. Postman also supports scripting for more advanced testing scenarios.
2. **SoapUI:** SoapUI is a tool primarily designed for testing SOAP (Simple Object Access Protocol) web services, but it can also be used for RESTful APIs. It offers both a free and a paid version, and it allows you to create comprehensive test suites with assertions and reporting.

What is meant by stub?

In software testing, a stub is a small, simple, and often incomplete piece of code or program that is used to simulate the behavior of a component or module that the code being tested depends on. Stubs are particularly useful in integration testing and unit testing when the components or modules being tested have dependencies on other components that may not yet be fully developed or available for testing.

What is meant by cause-effect graph testing?

The Cause-Effect Graphing technique is a systematic method used in software testing to design test cases based on the relationship between input conditions and their corresponding effects or outcomes. It is particularly useful for testing systems with complex logical conditions and rules. This technique helps testers create a structured and comprehensive set of test cases that cover various input scenarios.

Here's how the Cause-Effect Graphing technique works:

1. **Identify Inputs and Outputs:** Begin by identifying the inputs (causes) and outputs (effects) of the software or system under test. These can include various conditions, variables, and parameters that affect the behavior of the software.
2. **Create a Cause-Effect Graph:** Create a graphical representation, often in the form of a cause-and-effect graph or diagram. In this graph, each cause is represented as a node, and each effect is represented as an outcome or result.
3. **Define Relationships:** Connect the causes to their corresponding effects using logical relationships, such as "AND," "OR," and "NOT." These relationships indicate how different input conditions combine to produce specific outcomes.
4. **Reduce Complexity:** Simplify the graph by eliminating redundancy and unnecessary branches. The goal is to create a compact and manageable representation of the relationships between inputs and outputs.
5. **Generate Test Cases:** Use the cause-effect graph to derive test cases systematically. Each path through the graph represents a unique combination of input conditions that should be tested. Test cases are

generated to cover all the possible paths, ensuring comprehensive test coverage.

6. **Execute Tests:** Execute the generated test cases on the software or system, observing the actual outcomes. Compare the observed results with the expected results based on the graph to identify discrepancies and potential defects.