1. **What are the different types of testing?**

* Unit Testing –
  1. Unit testing is a method of testing individual units or components of a software application.
  2. It is typically done by developers and is used to ensure that the individual units of the software are working as intended.
* Integration Testing –
  1. Integration testing is a method of testing how different units or components of a software application interact with each other.
  2. It is used to identify and resolve any issues that may arise when different units of the software are combined.
* System Testing –
  1. This software is tested such that it works fine for the different operating systems.
  2. It is covered under the black box testing technique.
* Acceptance Testing –
  1. Acceptance testing is done by the customers to check whether the delivered products perform the desired tasks or not, as stated in requirements.
* Smoke Testing –
  1. This test is done to make sure that the software under testing is ready or stable for further testing.
  2. It is called a smoke test as the testing of an initial pass is done to check if it did not catch the fire or smoke in the initial switch on.
* Regression Testing –
  1. Regression testing is a method of testing that is used to ensure that changes made to the software do not introduce new bugs or cause existing functionality to break.
  2. It is typically done after changes have been made to the code, such as bug fixes or new features, and is used to verify that the software still works as intended.
* Performance Testing –
  1. It is designed to test the run-time performance of software within the context of an integrated system.
  2. It is used to test the speed and effectiveness of the program. It is also called load testing.
  3. In it we check, what is the performance of the system in the given load.
* Manual Testing –
  1. Manual testing is done in person, by clicking through the application or interacting with the software and APIs with the appropriate tooling.
* Automation Testing –
  1. Automated tests, on the other hand, are performed by a machine that executes a test script that was written in advance.
* Security Testing
* User Acceptance Testing
* Functional Testing

1. **What is Enzyme?**

* Enzyme is a JavaScript Testing utility for React that makes it easier to test your React Components' output.
* You can also manipulate, traverse, and in some ways simulate runtime given the output.

1. **Enzyme vs React Testing Library?**

* Enzyme –
  + Enzyme allows you to access the internal workings of your components.
  + You can read and set the state, and you can mock children to make tests run faster.
  + Enzyme is easier to grasp but in the long run, it's harder to maintain.
  + **Enzyme** is intended for unit/integration testing. Its API was designed to test the implementation
  + It offers custom renderer that doesn't require DOM (for shallow rendering), behaves differently from React renderer and allows things that are important for unit testing but aren't possible or straightforward with default renderer, like synchronous state updates, shallow rendering, disabling lifecycle methods, etc.
  + In Enzyme, we test the component using the state and props of the component. This generally means that the tests are brittle.
* React Testing Library –
  + react-testing-library doesn't give you any access to the implementation details.
  + It renders the components and provides utility methods to interact with them.
  + The idea is that you should communicate with your application in the same way a user would.
  + react-testing-library forces you to write tests that are a bit more complex on average but it rewards you with higher confidence in your code.
  + It uses React renderer and ReactTestUtils internally, requires real DOM because it's component's output that is asserted in tests, not internals.
  + It doesn't provide facilities for isolated unit tests but it's possible to do this by mocking modules that contain component that need to be spied, mocked or stubbed by other means, notably jest.mock.
  + Whereas in React Testing Library, we test the component from the user's perspective.
  + Users don't care what happens behind the scenes, they just see and interact with the output.
  + Instead of accessing the components' internal APIs or evaluating their state, you'll get more confidence by writing your tests based on the component output.

1. **What is Jest and why do we use it?**

* Jest is a delightful JavaScript Testing Framework designed to ensure correctness of any JavaScript codebase.
* It allows you to write tests with an approachable, familiar and feature-rich API that gives you results quickly.
* Jest is well-documented, requires little configuration and can be extended to match your requirements. Jest makes testing delightful.
* Jest is majorly used to work with react-native based web applications and with react, and it mostly focuses on simplicity while doing any unit testing.
* Also, the jest framework helps the programmer validate everything developed by using JavaScript, whether it is browser rendering of web applications or any mobile applications.
* For automated browser testing, Jest plays a very important role. Jest is among the most famous JavaScript testing frameworks currently compared with other frameworks in existence.
* It makes it easier and faster to run JavaScript tests by having all needed things included in one framework.
* It mainly aims to work out of the box, configuration free and on most JavaScript projects.
* It is very fast and safe; Jest can run all your tests parallel by ensuring they have a unique global state.
* To make things quicker and easier, it runs previously failed tests first and then reorganizes the runs of the next tests, which are based on the long test files.
* Jest has a great toolkit available to programmers at one place itself, which is very well documented, properly maintained and well arranged in a good manner.
* It also helps to maximize the performance level by managing the tests in a parallel manner by running them in their processes.
* Jest also maintained the information coverage records, and from entire projects and various untested files, it collects code coverage information.