***TestNG Interview Questions & Answer***

***What is the TestNG Framework?***

TestNG framework is a testing framework to perform tests in the java programming language. Moreover, the "***NG***" in TestNG abbreviates for "***Next Generation***". Cedric Beust developed it and inspired by the JUnit and NUnit testing framework.

***How do you run the TestNG script?***  
TestNG script is run by right-click on the ***TestNG class -> Run As -> TestNG Test***.

***What are the advantages of TestNG?***  
One of the common TestNG interview questions is about the advantages of TestNG.  
TestNG has the following advantages:

* *Firstly, TestNG is capable of producing reports automatically with all the necessary information* ***such as failed tests, passed tests, test execution times,*** *etc*.
* *TestNG makes use of annotations such as @BeforeMethod, @Test, etc., which are easily understandable as their naming is after their working*.
* *TestNG provides a grouping of methods by which we can* ***group multiple methods*** *as one unit. In other words, Grouping performs operations on all the tests in a group at once rather than individually*.
* *TestNG provides a test method parameterization, which means we can provide parameters in the TestNG and call the function repeatedly with different values. Moreover, parameterization helps in data-driven testing in TestNG*.
* *TestNG provides the prioritization of methods. In other words, by defining the priorities of the methods in TestNG, we can alter the default execution sequence of the test methods according to our wish*.
* *In addition to the above, TestNG allows parallel testing, which increases efficiency and improves the overall running time of test methods*.
* *With the TestNG framework, you can easily integrate with other tools such as Maven, Jenkins, etc*.
* *Moreover, TestNG provides a feature to run multiple*[***test methods on various browsers***](https://www.toolsqa.com/testng/cross-browser-testing-using-testng/)*to test for cross-browser compatibility issues on your website. It is*[***cross-browser testing***](https://www.toolsqa.com/cross-browser-testing/what-is-cross-browser-testing/).
* *Additionally, TestNG allows us to run the tests separately. So, if you run the tests and only one test failed, you can run this test independently in the next execution*.
* *Moreover, TestNG allows the test methods to depend on each other. Its also called Test Dependency in TestNG*.
* *Lastly, TestNG provides a bunch of assertion methods for testing more efficiently*.

**TestNG Test Case And Suites**

***What is the difference between a TestNG test and a TestNG test suite?***  
TestNG test suite refers to a collection of tests that we can run simultaneously with the help of the TestNG XML file. On the other side, a TestNG test is a single test case file, and when we say "***we are running a TestNG test case***", we simply mean we are running a single test case file.

***Define the correct order of tags in the TestNG XML file***.  
The correct order followed to run the TestNG suite from the XML file is as follows:

<suite>

<test>

<classes>

<class>

<methods>

The closing tags don't appear here as it is just for demonstration purposes.

**TestNG Annotations**

***What are the types of annotations used in TestNG (In the sequence of execution/hierarchy)?***

There are nine types of annotations used in TestNG. In order of their execution sequence, they are as follows:

* *@BeforeSuite*
* *@BeforeTest*
* *@BeforeClass*
* *@BeforeMethod*
* *@Test*
* *@AfterMethod*
* *@AfterClass*
* *@AfterTest*
* *@AfterSuite*

***What are the categories of annotations in TestNG?***

TestNG annotations divide into three categories:

* ***Precondition Annotations***: *The annotations under this category execute before the test. It consists of the following annotations*:
* *@BeforeMethod*
* *@BeforeClass*
* *@BeforeSuite*
* *@BeforeTest*
* ***Test Annotations***: *The annotations under this category are defined just before the test methods. Moreover, it consists of the following annotations*:
* *@Test*
* ***Postcondition Annotations***: *The annotations under this category execute after the test methods. Additionally, it consists of the following annotations*:
* *@AfterMethod*
* *@AfterClass*
* *@AfterTest*
* *@AfterSuite*

**TestNG Reports**

***What are the types of reports generated in TestNG by default?***

TestNG generates two types of reports by default after the execution of all the test methods finishes. They are:

* *Emailable Reports*
* *Index Reports*

***Where is the emailable report generated and saved in TestNG?***

Emailable reports generate under the project folder and test-output subfolder. This report is available as "***emailable-report.html***" by default.

***Where is the index report generated and saved in TestNG?***

The index report generates under the project folder and test-output subfolder. Moreover, this report is available as "***index.html***" by default.

**TestNG Priorities**

***What are priorities in TestNG?***

Priorities in TestNG is a parameter which declares the priority to a specific test method. TestNG uses the method of alphabetical execution to execute its test method. Through priorities, we can alter the sequence of the test execution. Additionally, the priority can be set as an integer value and lower this integer value; higher is the priority.

***How would you set priorities in TestNG?***  
TestNG priority is set by the following syntax:

@Test (priority = 1)

public void func(){

//test code

}

An example of prioritization in TestNG can be as follows:

@Test (priority = 1)

public void CloseBrowser() {

driver.close();

System.out.println("Closing Google Chrome browser");

}

@Test (priority = 0)

public void OpenBrowser() {

System.out.println("Launching Google Chrome browser");

driver.get("https://www.demoqa.com");

}

***Why do we create the XML file in TestNG?***

We use the XML file in TestNG for many purposes. The TestNG XML file helps us:

* *To run multiple tests in a single execution*.
* *Secondly, it also helps us to include and exclude the test methods and groups*.
* *Thirdly, it also helps us to add dependencies in groups*.
* *Fourthly, it helps to run the test case methods through parameters*.
* *Finally, it assists in the execution of the parallel test execution*.

**TestNG Parameters**

***What is parameterization in TestNG?***

In TestNG, parameterization means data driven testing runs a test method multiple times with different values. Another name for this process is data-driven testing in TestNG. We can acquire Parameterization in TestNG in two ways:

* *Firstly, we can achieve it through the XML file*.
* *Secondly, we can achieve it through the dataproviders in TestNG*.

***What are the optional parameters in TestNG?***

Optional parameters work similarly to the default case in the parameterization in TestNG. We use the optional parameter when no other parameter gets defined for that test case method. Additionally, the @Optional annotation declares the optional parameter. We don't define the @Optional parameter above the test method definition but alongside where the method is declared

import org.testng.annotations.Optional;

import org.testng.annotations.Parameters;

import org.testng.annotations.Test;

public class Params

{

@Test

@Parameters ("message")

public void OP( @Optional("Optional Parameter Selected") String message) {

System.out.println(message);

}

}

***Write the code snipped for passing values 1 and 2 to the parameters val1 and val2 through the XML file.***

To pass the values into the parameters in TestNG, we use <parameter> tag in the TestNG XML file. Additionally, it contains two attributes:

* *name: the name of the parameter variable.*
* *value: the value to insert in that variable.*

Observe the following XML file denoting the same concept.

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >

<suite name="My Test-Suite" >

<test name="QA" >

<parameter name="val1" value="1" />

<parameter name="val2" value="2" />

<classes>

<class name="testNGPackage.Parameter" />

<class name="testNGPackage.Multiple\_Parameters" />

</classes>

</test>

</suite>

**TestNG Groups**

***What is the importance of groups in TestNG?***

Another important TestNG interview questions are about its importance.

Groups are the collection of multiple test case methods combined into one single unit. By grouping, we can operate directly onto the group, which will reflect on all the test case methods under it. Moreover, in TestNG, we can also create a group of groups as a bigger unit of test methods.

***How do you define groups in TestNG?***

The answer to such TestNG interview questions is that we define the Groups in TestNG by passing the "***groups***" parameter to the Test annotation with the value being the group name. In the below example, the test case method will be under the group named "***group1***".

***@Test ( groups = {"group1"})***  
***//test case method***

***How do you exclude a group from the test execution cycle?***

Excluding a group in TestNG denotes that this particular group refrains from running during the execution, and TestNG will ignore it. Additionally, the name of the group that we want to exclude is defined in the XML file  by the following syntax:

<groups>

<run>

<exclude name = "demo">

</exclude>

</run>

</groups>

***Can we use regular expression in TestNG groups? Write a demo XML file for the same.***

Yes, regular expressions can be used in TestNG to execute the groups which have some typical pattern in their name.

For example, if I want to run all the groups with a name starting from "***ABC***", then I can write the regular expression as abc.\* in the XML file.

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >

<suite name="Test-Suite" >

<test name="ToolsQA" >

<groups>

<run>

<include name = "abc.\*">

</include>

</run>

</groups>

<classes>

<class name="TestNG" />

</classes>

</test>

</suite>

**TestNG Asserts**

***What do you understand by asserts in TestNG?***

An asset is a piece of code that helps us verify if the expected result and the actual result are equal or not. In TestNG, we leverage the inbuilt "***Assert***" class and a lot of its method to determine whether the test case passed or failed. Additionally, in TestNG, a test case acts as a "***pass***" if none of the assert methods throws an exception during the execution. The syntax for TestNG assert is: Assert.Method(actual, expected, message);

***Describe any five common TestNG assertions.***

The five common TestNG assertions are:

* *assertEqual(String actual,String expected)*
* *assertEqual(String actual,String expected, String message)*
* *assertEquals(boolean actual,boolean expected)*
* *assertTrue(condition)*
* *assertTrue(condition, message)*
* *assertFalse(condition)*
* *assertFalse(condition, message)*

***What are the different types of assert in TestNG?***

There are two types of asserts in TestNG:

* *Soft Asserts*
* *Hard Asserts*

***Define soft asserts in TestNG and describe how they are different from hard assert.***

Soft asserts in TestNG means that the execution of the tests would not stop even though the assertion throws an exception in between the execution. In addition to this, TestNG does not include Soft asserts by default in TestNG, so an extra ***org.testng.asserts.Softassert*** package import is required.

Moreover**, Soft asserts are different from hard asserts as the hard asserts stop the execution of the test case as soon as the first assertion fails and provides the results. Hard assert includes by default in TestNG.**

**TestNG Dependent Tests**

***What is meant by dependency in TestNG?***

Dependency in TestNG is a process of making one test dependent on the other test. By providing dependencies in the test methods, we assure that a test method B would only run if test method A runs (*given B depends on A*). Moreover, in TestNG, we can also have one test method dependent on multiple tests.

***How do you create dependencies in TestNG?***

We can create the dependent tests in TestNG by providing the *dependsonMethods* parameter on the @Test annotation. The value of the attribute is the name of the method on which we want this method to depend. The usage of this method is as follows:

import org.testng.annotations.Test;

public class DependsOnTest {

@Test (dependsOnMethods = { "OpenBrowser" })

public void SignIn() {

System.out.println("User has signed in successfully");

}

@Test

public void OpenBrowser() {

System.out.println("The browser is opened");

}

}

***How do you create dependency through the XML file?***

TestNG also allows us to create dependencies between groups through the TestNG XML file. Such dependencies denote the dependence of one group onto another. The following code demonstrates how to achieve the same goal:

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >

<suite name="TestNG XML Dependency Suite" >

<test name="ToolsQA" >

<groups>

<dependencies>

<group depends-on= "openbrowser" name= "login"></group>

</dependencies>

</groups>

<classes>

<class name="GroupDependency" />

</classes>

</test>

</suite>

Here, the login group depends upon the *openbrowser* group.

***When do we use "dependsOnGroups" in TestNG?***

TestNG gives us the liberty to make a single test depend on a group of tests. When we want to execute in such a manner, we use the *dependsOnGroups* attribute in the TestNG test case file. The value of this attribute is the name of the group on which we want this method to depend. Given below is an example demonstrating the same:

import org.testng.annotations.Test;

public class GroupDependency

{

@Test(dependsOnGroups = { "SignIn" })

public void ViewAcc() {

System.out.println("View Your Dashboardd");

}

@Test(groups = { "SignIn" })

public void OpenBrowser() {

System.out.println("Browser Opened Successfully");

}

@Test(groups = { "SignIn" })

public void LogIn() {

System.out.println("Login Into The Account");

}

}

**Miscellaneous**

***What is the significance of "timeout" in TestNG?***

In TestNG, "***timeout***" is a parameter that defines the maximum time that a method can take for execution. The timeout parameter is handy if the execution time is a constraint during testing, or the tester wants to take precautions against the endless execution of tests. We can declare the timeout parameter at:

* ***suite level***: *To put a time constraint on all the methods in the suite*.
* ***method level***: *To put the time constraint on a particular method*.

Syntax: *@Test (timeout = 1000)*

***What is meant by invocationCount in TestNG?***

The *invocationCount* is an attribute that defines the number of times a test method has to run in a single execution. So, if the *invocationCount* sets as 5, then the test method would run five times each time I execute the TestNG test case.

Syntax for invocationCount: *@Test (invocationCount = 5)*

***What is meant by parallel test execution in TestNG?***

The parallel test execution means executing different test methods simultaneously, i.e., parallelly in TestNG. It is achieved by creating threads and assigning these threads to different test methods (*which is done automatically and is an operating system's job*). Moreover, running the tests parallelly rather than sequentially is very efficient.

***On what levels can we apply parallel testing in TestNG?***

Parallel testing can apply at four different levels in TestNG:

* ***Methods***: *This will run the parallel tests on all @Test methods in TestNG*.
* ***Tests***: *All the test cases present inside the <test> tag will run with this value*.
* ***Classes***: *All the test cases present inside the classes that exist in the XML will run in parallel*.
* ***Instances***: *This value will run all the test cases parallelly inside the same instance*.

***How is exception handling done in TestNG?***

We carry out Exception handling in TestNG by defining the exception at the @Test annotation level. If we proceed in such a manner, the test case will not fail even after raising an exception.

*@Test (expectedException = numberFormatException.class)*

A tester can write any type of exception here instead of *numberFormatException*.

***Can we disable a test in TestNG? If so, explain how?***

Yes, disabling a test can be achieved in TestNG. Once we disable a test, it will not run in the next execution cycle. Moreover, we accomplish this by using the "***enabled***" attribute**. Syntax:** *@Test (enabled = False)*

***Why is the reporter class used in TestNG?***

The reporter class in TestNG logs the tester defined messages into the reports generated by TestNG. These logged messages then print into the reports, which we can share with the team.

***Define the syntax for generating logs through the reporter class in TestNG.***

Reporter class logs tester-defined messages onto the reports generated by TestNG. Additionally, the syntax for the same is as follows: *Reporter.log("message");*

***What is @Factory annotation in TestNG?***

The need to run multiple test cases in a single test suffices by using the @Factory annotation. The name factory resembles the generation of test class object that is provided by the method under it.

Moreover, it is similar to a factory producing a product.

@Factory()

public Object[] getTestClasses() {

Object[] tests = new Object[2];

tests[0] = new Test1();

tests[1] = new Test2();

return tests;

}

***Note***: *The test method under @Factory annotation always returns an object array.*

***What is the difference between @Factory and @Dataprovider annotations?***

*@Factory* and *@Dataprovider* are two types of annotations available in TestNG, which look similar in their working but are different.

***@Dataprovider*: *The dataprovider annotation enables the tester to run a test method multiple times using a different set of data provided by the dataprovider*.**

***@Factory***: ***The use of the factory annotation is when the tester needs to execute the test methods multiple times, which are present in the same class. Additionally, we achieve this by creating different instances of the same class*.**

**public class SimpleTest {**

**@Test**

**public void simpleTest() {**

**System.out.println("Simple Test Method.");**

**}**

**}**

**public class SimpleTestFactory {**

**@Factory**

**public Object[] factoryMethod() {**

**return new Object[] { new SimpleTest(), new SimpleTest() };**

**}**

**}**

* **In this example:**
  + The SimpleTestFactory class defines a factory method annotated with @Factory.
  + The factory method returns two instances of the SimpleTest class.
  + TestNG executes the test method from SimpleTest twice.

**TestNG Listeners**

***What are listeners in TestNG?***

TestNG listeners allow you to customize and enhance your test execution process. Listeners in TestNG are the piece of code that listens to certain events and execute.

**In TestNG**, listeners are Java classes that implement specific interfaces defined by TestNG. These interfaces define methods that get called before or after specific events in the test lifecycle.

**ITestListener: (**its an interface)

* + Monitors the start and completion of test methods.
  + Key methods:
    - onTestStart(ITestResult result): Invoked when any test method starts.
    - onTestSuccess(ITestResult result): Executed on the success of a test method.
    - onTestFailure(ITestResult result): Invoked when a test method fails.
    - onTestSkipped(ITestResult result): Runs only when any test method is skipped.
    - onTestFailedButWithinSuccessPercentage(ITestResult result): Called when a test method fails but within the configured success percentage.
    - onStart(ITestContext context): Executed at the start of any test method.
    - onFinish(ITestContext context): Invoked when any test case finishes its execution.

**ISuiteListener: (**its an interface)

* + Manages suite-level events.
  + Methods:
    - onStart(ISuite suite): Called when a suite starts.
    - onFinish(ISuite suite): Invoked when a suite finishes.

1. **IReporter:** (its an interface)
   * Generates customized reports post-test execution.
   * Method:
     + generateReport(List<XmlSuite> xmlSuites, List<ISuite> suites, String outputDirectory): Generates the report.

***How are listeners declared in TestNG?***

The listener code in TestNG exists in a separate file than the TestNG test case file. Subsequently, this file contains the listener code and the type of listener to implement is done by "***implementing***" the listener class in the following way:

public class ListenersTestNG implements ITestListener {

public void onStart(ITestContext context) {

System.out.println("onStart method started");

}

}

To apprise the TestNG test case file about the listener, we declare the @Listener annotation and mentioning the listener class name in the following manner:

**@Listeners(ListenersTestNG.class)**

public class TestNG {

WebDriver driver = new FirefoxDriver();

@Test //Success Test

public void CloseBrowser() {

driver.close();

}

}

***What do we need to generate a customized report in TestNG?***

A customized report in TestNG generates with the help of TestNG listeners. Using the interface *ITestListener* in TestNG, we can control the events such as method start, method pass, fail, etc., and according to these events, a tester can log appropriate messages.

***The sequence of execution of all the annotations in TestNG is given below:***

* @BeforeSuite
* @BeforeTest
* @BeforeClass
* @BeforeMethod
* @Test
* @AfterSuite
* @AfterTest
* @AfterClass
* @AfterMethod

***What SoapUI can do?***

* SoapUI is a powerful tool for testing web services.
* It allows you to:
* Perform functional testing of SOAP and RESTful web services.
* Conduct performance **testing, interoperability testing, and regression testing.**
* Create and execute automated tests.
* Simulate web services.
* Record and reuse tests.
* Generate code stubs from WSDL.
* Create REST specifications (WADL) from recorded communication.

|  |  |
| --- | --- |
| **WADL** | **WSDL** |
| Web Application Description Language. | Web Service Description Language. |
| XML helps in describing restful web services. | XML helps in describing SOAP-based web series. |
| A client loads a WADL file and accesses the complete functionality of the web services. | A client loads the WSDL file and gets into the method web services which call the arguments and returns data type. |
| WADL is equivalent to SOAP’s web services description language. | WSDL is a machine-readable description with the current version of 2.0. |
| It is lightweight, easy to understand and write as compared to WSDL. | Difficult to understand and write as compared to WADL. |
| It does not evolve binding to SMTP servers. It has a simple design. | It consists of binding to STMP services. it consists of complex design. |
| It requires some simple URI template mechanism. Also, limited scope compared to WSDL. | Whereas, here the users need to define the XML input message to use the URI template mechanism.  It has more range compared to WADL. |
| W3C does not recommend the WADL interface. WADL is very simple. | But W3C does recommend a WSDL interface. whereas, it is more flexible than WADL. |

***What is web services ?***

A web service is a set of open protocols and standards that allow data to be exchanged between different applications or systems.

Any software, application, or cloud technology that uses standardized web protocols (HTTP or HTTPS) to connect, interoperate, and exchange data messages – commonly XML (Extensible Markup Language) – across the internet is considered a web service.

|  |  |
| --- | --- |
| **HTTP** | **HTTPS** |
| The full form of HTTP is **the Hypertext Transfer Protocol.** | The full form of HTTPS **is Hypertext Transfer Protocol Secure.** |
| It is written in the address bar as http://. | It is written in the address bar as https://. |
| The HTTP transmits the data over port number 80. | The HTTPS transmits the data over port number 443. |
| It is unsecured as the plain text is sent, which can be accessible by the hackers. | It is secure as it sends the encrypted data which hackers cannot understand. |
| It is mainly used for those websites that provide information like blog writing. | It is a secure protocol, so it is used for those websites that require to transmit the **bank account details or credit card numbers.** |
| **It is an application layer protocol.** | **It is a transport layer protocol.** |
| It does not use SSL. | It uses SSL that provides the encryption of the data. |
| Google does not give the preference to the HTTP websites. | Google gives preferences to the HTTPS as HTTPS websites are secure websites. |
| The page loading speed is fast. | The page loading speed is slow as compared to HTTP because of the additional feature that it supports, i.e., security. |

***Default Properties in SOAPUI?***

When you create a new project in SoapUI, it comes with some default properties. These properties include information **like endpoints, authentication credentials**, **and session IDs.** You can view and manage these properties in the Project Navigator.

* **An endpoint** is a specific URL (web address) where clients can access a particular web service.
* Think of it as the entry point to interact with a specific functionality provided by the service.

***Important Functionalities of SOAP UI:***

* **User-Friendly GUI:** SoapUI’s interface is intuitive, even for new users. Creating projects, adding test suites, and managing assignments are straightforward.
* **Functional Testing:** Drag-and-drop options allow you to create test scenarios without writing background scripts. You can debug tests and perform data-driven testing.
* **Vulnerability Testing:** Protect applications by executing methods like Test Generator, SQL Injection, and XML Bomb.
* **Load Testing using LoadUI:** Assess how your web service handles load and stress.
* **Automation with Groovy:** Write custom scripts for advanced scenarios.
* **Data-Driven Testing:** Validate responses with different data inputs.
* **Assertions:** Checkpoints to validate response content, compliance, and more.

***What is SOAP UI?***

SOAP UI is an open-source cross-platform API testing tool used for testing web services. Developed by Eviware in 2005 and later acquired by SmartBear, it allows testers to execute automated functional, regression, compliance, and load tests on various web APIs.

***Role of XML, SOAP, WSDL, and UDDI in Web Services:***

* **XML (eXtensible Markup Language):** Used to structure data in web service requests and responses.
* **SOAP (Simple Object Access Protocol**): An XML-based messaging protocol for exchanging structured information between applications.
* **WSDL (Web Services Description Language):** Describes web services, their operations, inputs, and outputs. Developers can understand how to interact with a service without accessing its source code.
* **UDDI (Universal Description, Discovery, and Integration):** Facilitates service discovery and integration. It helps list accessible services.

***Assertions in Soap UI:***

* Assertions validate the response received by a test step during execution. Commonly used types include:
* **Contains Assertion:** Checks if a specified string exists in the response.
* **Not Contains Assertion:** Verifies the non-existence of a specified string.
* **Xpath Match Assertion:** Validates response content using XPath expressions.
* **Script Assertion:** Allows custom validation using Groovy scripts.
* **Security assertions**

***Understanding Web Services:***

* Web services enable different applications to communicate and exchange data over the internet. They rely on standardized protocols like SOAP, WSDL, and UDDI.
* Communication Channels for Web Services:
* HTTP/HTTPS: Commonly used for RESTful web services.
* JMS (Java Message Service): For asynchronous communication.
* SMTP (Simple Mail Transfer Protocol): For email-based services.
* FTP (File Transfer Protocol): For file exchange.
* MQTT (Message Queuing Telemetry Transport): Lightweight protocol for IoT devices.

***SoapUI Automation:***

SoapUI allows you to automate test execution, integrate with CI/CD pipelines, and schedule test runs. You can use Groovy scripts for advanced automation scenarios.

***Difference between TestNG and cucumber?***

***Programming Language:–***

*TestNG: Primarily designed for Java, but offers support for multiple languages through various plugins.– Cucumber: Supports multiple programming languages, including Java, Ruby, and JavaScript, making it versatile for diverse development environments.*

***Support for Development:***

*– TestNG: Well-suited for developers, especially in Java-centric environments, providing annotations for test configuration.*

*– Cucumber: Geared towards collaboration between developers and non-technical stakeholders, as it uses Gherkin syntax for writing tests in plain language.*

*Type of Testing:*

*– TestNG: Widely used for unit, integration, and end-to-end testing.*

*– Cucumber: Primarily focuses on behavior-driven development (BDD) and acceptance testing, allowing tests to be written in natural language.*

*Use Case:*

*– TestNG: Ideal for testing a broad range of applications and scenarios, offering flexibility in test design.*

*– Cucumber: Best suited for scenarios where collaboration between technical and non-technical team members is crucial, promoting clear communication through feature files.*

***xUnit:***

*– TestNG: Follows the xUnit architecture and provides annotations for test lifecycle management.*

*– Cucumber: Has its unique Gherkin syntax, deviating from the traditional xUnit structure, emphasizing behavior-driven development.*

***Server-Side:***

*– TestNG: Well-suited for server-side testing, ensuring comprehensive coverage in application testing.*

*– Cucumber: Can be applied to server-side testing, focusing on behavior-driven development principles.*

*Grouping:*

*– TestNG: Provides flexible grouping mechanisms, allowing the execution of specific test groups.*

*– Cucumber: Uses feature files and scenarios for organization, but grouping is less explicit than in TestNG.*

***Generators:***

*– TestNG: Offers data-driven testing capabilities, allowing the use of data providers for dynamic test case generation.*

*– Cucumber: Excels in data-driven testing through scenarios and tables in feature files, providing a clear structure for test generation.*

*In comparing Cucumber and TestNG, the choice depends on the project’s nature, team composition, and the emphasis on behavior-driven development principles. While TestNG is versatile and well-suited for various testing scenarios, Cucumber excels in scenarios where collaboration and communication through natural language are pivotal.*

1. ***What is Cucumber? Why is it used?***
   * ***Cucumber****is a testing tool based on the****Behavior Driven Development (BDD)****framework.*
   * *It allows running functional tests written in plain text and developing test cases for software functionality.*
   * [*Cucumber plays a supporting role in automated testing, making it easier to collaborate between developers, testers, and business analysts in Agile software development approaches*](https://www.lambdatest.com/learning-hub/cucumber-interview-questions)
2. ***Advantages of Using Cucumber:***
   * ***Plain-Text English****: Cucumber tests are written in plain-text English, enabling collaboration with stakeholders who have minimal technical knowledge.*
   * ***Code Reusability****: High code reuse in tests.*
   * [***Integration:*** *Compatible with Selenium and various testing frameworks (e.g., JUnit, TestNG*](https://www.lambdatest.com/learning-hub/cucumber-interview-questions)
3. ***What Types of Testing Can Be Performed Using Cucumber?***
   * *Cucumber supports various types of testing, including:*
     + ***Acceptance Testing****: Evaluating software performance and compliance with requirements.*
     + ***Functional Testing****: Verifying software functionality.*
     + [***Regression Testing****: Ensuring existing functionality remains intact*](https://www.lambdatest.com/learning-hub/cucumber-interview-questions)[*1*](https://www.lambdatest.com/learning-hub/cucumber-interview-questions)*.*
4. ***Gherkin Syntax and BDD Principle:***
   * ***Gherkin****is the language used by Cucumber.*
   * *It provides a simple English representation of application behavior.*
   * [*Keywords like****Feature****,****Scenario****,****Given****,****When****, and****Then****describe application behavior*](https://www.lambdatest.com/learning-hub/cucumber-interview-questions)[*1*](https://www.lambdatest.com/learning-hub/cucumber-interview-questions)*.*
5. ***Files Required for Cucumber Test Execution:***
   * ***Feature File****: Provides a high-level description of the Application Under Test (AUT).*
   * [***Step Definition****: Contains the actual code implementation corresponding to feature scenarios*](https://www.lambdatest.com/learning-hub/cucumber-interview-questions)*.*
6. ***Background Keyword in Cucumber:***
   * *Used to group multiple****Given****statements into a single group.*
   * [*Useful when the same set of****Given****statements repeats across scenarios in the feature file1*](https://www.lambdatest.com/learning-hub/cucumber-interview-questions)*.*
7. ***Scenario Outline Usage:***
   * *Scenario Outline allows executing the same scenario with multiple sets of data.*
   * *The test steps remain the same, but parameter values change.*
   * [*Followed by the****Examples****keyword, specifying different parameter values*](https://www.lambdatest.com/learning-hub/cucumber-interview-questions)[*1*](https://www.lambdatest.com/learning-hub/cucumber-interview-questions)*.*
8. ***Step Definitions:***
   * [*Actual code implementation of features mentioned in the feature file*](https://www.lambdatest.com/learning-hub/cucumber-interview-questions)[*1*](https://www.lambdatest.com/learning-hub/cucumber-interview-questions)*.*
9. ***Supported Programming Languages:***
   * [*Cucumber supports multiple languages, including****Java****,****.Net****, and****Ruby***](https://www.lambdatest.com/learning-hub/cucumber-interview-questions)
10. ***Selenium vs Cucumber***
    * ***Cucumber:***
      + *Behavior-Driven Development (BDD) tool.*
      + *Develops test cases based on the behavior of software functionality.*
      + *Supports collaboration between testers, developers, and business analysts.*
      + *Test cases are written in plain English using the Gherkin language.*
    * ***Selenium:***
      + *Web UI automation testing suite.*
      + *Primarily focuses on automating user interface (UI) testing.*
      + *Widely used for testing web applications across different browsers and platforms.*
11. ***Type of Testing:***
    * ***Cucumber:***
      + ***Acceptance testing. Functional testing***
      + ***Validates software behavior against requirements.***
    * ***Selenium:***
      + ***UI testing, Functional , Integration , Acceptance , Performance***
      + ***Ensures the correctness of web application interfaces.***
12. ***Script Creation:***
    * ***Cucumber:***
      + ***Simpler script creation.***
      + ***Test cases written in plain text (Gherkin language).***
    * ***Selenium:***
      + ***Requires more technical expertise.***
      + ***Test scripts coded in programming languages (e.g., Java, C#).***
13. ***Supported Languages:***
    * ***Cucumber:***
      + ***Supports various languages beyond Ruby (e.g., Java, Scala, Groovy).***
    * ***Selenium:***
      + ***Supports multiple languages (e.g., Java, .Net, Python, Ruby).***
14. ***Collaboration:***
    * ***Cucumber:***
      + ***Involves both testers and developers in writing automation steps.***
    * ***Selenium:***
      + ***Also includes both testers and developers in the automation process.***
15. ***Functional vs. Performance Testing:***
    * ***Cucumber:***
      + ***Functional testing.***
    * ***Selenium:***
      + ***Both functional and performance testing (via Selenium Grid).***

***Difference between findElement and initElements?***

* *Purpose: The findElement method is used to locate a single web element on a web page.*
* *Usage: You call findElement on a WebDriver instance (usually representing a browser session) to search for an element based on a specified locator strategy (such as by ID, name, CSS selector, or XPath)*
* *Usage: It initializes the web elements defined in a page object class (usually a subclass of BasePage or a specific page) using the provided WebDriver instance*
* *When you call initElements(driver, this) within a page object constructor, it scans the class for @FindBy annotations (or other supported annotations) and initializes the corresponding web elements.*
* *This allows you to directly interact with the web elements without explicitly using findElement calls in your test code.*
* @FindBy(id = "username")

***How many ways can we execute test in a specific order ?***

* *Priority attribute usage*
* *Using the dependsOnMethods attribute*
* *Using the dependsOnGroups property*
* *Making use of an XML configuration file*

***Order to run TestNG suite from the XML?***

*Suite>listener>test>parameter>classes>class>method>include>exclude>groups>run>include*

***How to use regular expressions in TestNG XML file ?***

<**run**>

<**include** name=".\*smoke.\*"/>

</**run**>

***What is the use of preserve order attribute in the TestNG XML file ?***

***It is used to run method in defined order.****If the preserve-order property is set to true, TestNG will keep the order of test methods as defined in the Java code. TestNG may run the test methods in any order if preserve-order="false."*

***How to integerate TestNG XML with maven ?***

*Add the TestNG dependency to your Maven pom.xml file, create a TestNG XML file and add your test classes and configurations, add the maven-surefire-plugin to the build section of your pom.xml file, and run your Maven tests using the test command to integrate TestNG XML with Maven.*

***How to pass the parameter in the test case through the testng.xml file?***

You can use the parameter tag within the test tag to pass parameters in the test case via the TestNG XML file. First, You need to specify the parameter name and value within the tag. These parameters can be accessed in the test case by using the @Parameters annotation in the method signature, as shown below:

@Test

@Parameters({"username", "password"})

public void myTestMethod(String username, String password) {

//test method code

}

***What are the categories of annotations in TestNG?***

***Precondition Annotations:*** ***@BeforeMethod, @BeforeClass, @BeforeSuite, @BeforeTest***

***Test Annotations: @Test***

***Postcondition Annotations: @AfterMethod, @AfterClass, @AfterTest, @AfterSuite***

***Meta Annotation: @TestInstance, @Parameters, @DataProvider, @Listeners, @Factory,@Test***

***What is @Factory annotation in TestNG?***

In TestNG, the @Factory annotation is used to create test instances at runtime. It enables you to generate test classes or instances dynamically based on runtime conditions or parameters.

You can use **the @Factory annotation to create multiple instances of the same test class with different data sets or parameters and run them in parallel to improve test execution speed.**

@Factory

public static Object**[]** createInstances() {

Object**[]** result = new Object[3];

for (int i = 0; i < result.length; i++) {

result**[i]** = new MyTest(i);

}

return result;

}

### What is the difference between @Factory and @Dataprovider annotations?

The **@Factory** annotation is used to create multiple instances of the same test class, each with a different set of input parameters or configurations. This is useful when running the same test with different data sets or configurations or when running tests in parallel to save time during testing.

Meanwhile, **@DataProvider** annotation supply test data to a test method. It allows you to separate the test data from the test logic, making your test code easier to maintain and reuse. @DataProvider can be used to provide test data from a variety of sources, including arrays, Excel sheets, databases, and external files.

### What is the use of @Listener annotation in TestNG?

In TestNG, the @Listener annotation is used to define listeners, which are classes that listen to events that occur during the execution of a TestNG test. Listeners can be used to customize or enhance TestNG's behavior by adding extra functionality such as logging, reporting, or customizing the test execution flow.

You can specify one or more listener classes to be invoked during test execution by using the @Listener annotation. This enables you to extend and customize TestNG's behavior to meet your specific testing requirements.

### ***What are the attributes supported by @Test annotation in TestNG?***

The following is an exhaustive list of the attributes supported by the @Test annotation in TestNG:

* alwaysRun
* dataProvider
* dataProviderClass
* description
* enabled
* expectedExceptions
* expectedExceptionsMessageRegExp
* groups
* invocationCount
* invocationTimeOut
* priority
* successPercentage
* testName
* timeOut
* dependsOnGroups
* dependsOnMethods
* skipFailedInvocations
* alwaysResolveDependencies

### 27. Which attribute is used to run the test method always?

The **alwaysRun** attribute ensures that a test method is always executed, regardless of whether a dependent method fails or is skipped.

### 28. Which attribute is used to provide data to test method in Data-driven testing?

In Data-driven testing, we use the **dataProvider** attribute of the @Test annotation in TestNG to provide data to a test method.

### 29. What is the use of the dependsOnMethods attribute in TestNG?

The dependsOnMethods attribute in TestNG is used to specify a test method's dependency on one or more other test methods. This property allows us to ensure that certain test methods are run in the correct order.

When one test method is dependent on another, TestNG will not run the dependent method until all of its dependencies have passed successfully. The dependent method will be skipped if any of the dependencies fail or are skipped.

### 30. What is the use of dependsOnGroups attribute in TestNG?

The **dependsOnGroups** attribute in TestNG is used to specify a test method's dependency on one or more groups of test methods. This property allows us to ensure that certain groups of test methods are executed in the correct order.

When a test method is dependent on a group of test methods, TestNG will not execute the dependent method until all of the test methods in the specified group have successfully passed. If any of the group's test methods fail or are skipped, the dependent method will also be skipped.

### What is the difference between @BeforeTest and @BeforeMethod annotation?

The @BeforeTest annotation is used to run the setup code before running all of the test methods in the <test> tag of a TestNG XML file. The @BeforeMethod annotation, on the other hand, is used to run the setup code before each individual test method in a class.

### 32. How will you make test cases dependent on each other?

Using the @Test annotation's **dependsOnMethods** attribute, we can make test cases dependent on one another in TestNG. This attribute allows us to specify which test methods must be run before a specific test method can be executed.

33. What are the types of reports generated in TestNG by default?

* TestNG HTML reports
* TestNG XML reports
* TestNG index reports
* Emailable reports

Apart from these, the TestNG Listener API allows you to create custom reports. Custom listeners can be created by developers to generate reports in a variety of formats, including JSON, PDF, and CSV.

### 34. What are the different ways to produce reports for TestNG results?

TestNG provides several methods for producing test result reports. Here are some common methods for [generating reports for TestNG](https://www.lambdatest.com/blog/how-to-generate-testng-reports-in-jenkins/) results:

* **TestNG Default Reports:** For test results, TestNG provides default HTML reports. These reports include detailed information about the tests, such as test methods, groups, execution time, and test status.
* **TestNG Ant Task:** TestNG includes an Ant task for creating customized HTML reports. You can use this task to create customized reports with logos, headers, and footers.
* **ExtentReports:** ExtentReports is a popular reporting library that generates rich, interactive HTML reports for TestNG results. It has features such as pie charts, line charts, and screenshots.
* **Allure Reports:** Another popular reporting library that provides detailed and interactive HTML reports for TestNG results is Allure Reports. It includes features such as history, tags, and attachments.
* **ReportNG:** ReportNG is a TestNG custom reporting plugin that generates detailed HTML reports with customizable templates. It includes features such as screenshots, pie charts, and test logs.

### 35. Where is the emailable report generated and saved in TestNG?

The emailable report is generated by default in the TestNG project's output directory, in the folder named "**test-output**." The emailable report's filename is "**emailable-report.html**."

<**project**-dir>/test-output/emailable-report.html

### 36. Where is the index report generated and saved in TestNG?

### The index.html report is one of TestNG's default reports, providing an overview of the test results as well as links to other reports such as the emailable report, **testng-results.xml**, and **testng-failed.xml**.

The index.html report is generated by default in the TestNG project's output directory under the folder named "**test-output**." The index.html report's filename is "**index.html**."

<**project**-dir>/test-output/index.html

It should be noted that the output directory can be specified in the TestNG XML file by using the "outputDirectory" attribute of the "suite" element.

### 37. How to use TestNG Reporter Class for the log generation?

The TestNG Reporter class is a built-in class for logging messages during test execution. Follow these steps to use the Reporter class for log generation:

* Import the Reporter class as follows:

**import** org.testng.Reporter;

* As an argument, pass a message string to the Reporter.log() method:

Reporter.log("Log message");

* As a second argument to the log() method, you can optionally include a log level, such as "**debug", "info", "warning", or "error":**
* To view the Reporter class's logs, enable the verbose output option in the TestNG XML file:

<suite name="My Test Suite" verbose="1"></suite>

When the verbose output option is enabled, the [Reporter class logs](https://www.lambdatest.com/blog/how-to-use-testng-reporter-log-in-selenium/) are displayed in the console and included in TestNG reports.

### 38. What do we need to generate a customized report in TestNg

In TestNG, you must use a combination of listeners, reporters, and frameworks to generate a customized report. You can take the following steps:

* Make use of the TestNG IReporter interface.
* Implement the **ITestListener interface in TestNG.**
* Include the listeners and reporters in your testing.
* Run your test suite and look at your customized report.

### 39. How to share the project report using TestNG?

You can distribute the project report generated by TestNG in the following ways:

* **HTML Reports:** You can distribute these reports to others by simply sending the HTML file or hosting the file on a server and sending the link.
* **Email Reports:** TestNG also allows you to generate and send an email report with a summary of the test execution results. In the testng.xml file, you can configure the email settings, including the email recipients, subject, and body.
* **CI/CD tools:** Integration with Continuous Integration/Continuous Deployment (CI/CD) tools: If you use a CI/CD tool like Jenkins or Bamboo, you can integrate TestNG with these tools to automatically generate and share test execution reports.

## Grouping related TestNG interview questions

### 40. What is the importance of groups in TestNG?

TestNG groups enable you to categorize your tests and control their execution based on these categories. By grouping your tests, you can easily select which to run and which to skip based on the group to which they belong.

Groups have several advantages, such as:

* You can organize your tests however you see fit, such as by functionality, priority, or environment.
* Run a subset of the tests in your test suite rather than all of them, which can save you time and effort.
* Test execution in different groups concurrently, which allows for faster execution and better test coverage.
* managing and organizing your tests according to groups, making it easier to maintain and update your test suite.

### 41. **Define grouping in TestNG?**

The process of categorizing test methods into logical groups based on their functionality or purpose is referred to as grouping in TestNG. We can execute a subset of test methods or all test methods in a group by grouping. We can define groups at several levels, including class, package, and suite.

### 42. How do you define groups in TestNG?

Groups are a collection of multiple test case methods that have been combined into a single unit. We can operate directly on the group by grouping, which will reflect on all of the test case methods under it. Furthermore, in TestNG, we can define a group of groups as a larger unit of test methods.

### 43. How do you exclude a group from the test execution cycle?

The exclusion a group in TestNG indicates that this group will not run throughout the execution and will be ignored by TestNG. Furthermore, the name of the group to be excluded is defined in the XML file using the following syntax:

<**groups**>

<**run**>

<**exclude** name = "TestNG testing">

</**exclude**>

</**run**>

</**groups**>

### 44. How to create a Group of Groups in TestNG?

You can make a group of groups in TestNG by including one or more groups as dependencies on another group. The following are the steps for creating a group of groups in TestNG:

* In your test class, use the @Test annotation with the groups attribute to define groups. As an example:

@Test(groups = {"group1"})

public void testMethod1() {

// test code

}

@Test(groups = {"group2"})

public void testMethod2() {

// test code

}

@Test(groups = {"group3"})

public void testMethod3() {

// test code

}

* Make a new group with the groups you want to group together as dependencies. As an example:

@Test(groups = {"groupOfGroups"}, dependsOnGroups = {"group1", "group2", "group3"})

**public** void **testGroupOfGroups**() {

// test code

}

In the preceding example, the testGroupOfGroups method is a member of the groupOfGroups group and is dependent on the groups group1, group2, and group3.

* Use TestNG to run your tests. The testGroupOfGroups method will only be called if all of the tests in groups 1, 2, and 3 passes.

### **45. How to group multiple test methods in a single group using TestNG?**

With the @Test annotation's groups feature, you can organize many test methods into a single group in TestNG. Here's how to use TestNG to group many test methods into a single group:

**Syntex:**

@Test(**groups** = {"GroupName"})

### **46. How do group multiple test methods in multiple groups?**

With the @Test annotation's groups feature, you can group multiple test methods into multiple groups in TestNG.

To divide a test method into numerous groups, use the groups attribute to give a comma-separated list of group names. As an example:

@Test(groups = {"Science", "Maths"})

**public** void **college**() {

// test code

}

The college method is included in both Science and Maths in the preceding example. Similarly, you can organize numerous test methods into several groups by using the groups attribute to define the same group names for each method.

### **47. How to group multiple test methods with Priority?**

The priority attribute of the **@Test** annotation in TestNG allows you to group multiple test methods with different priorities. The **priority** attribute specifies the order in which TestNG should execute the test methods.

To divide a test method into numerous groups, use the groups attribute to give a comma-separated list of group names. As an example:

@Test(priority = 1, groups = {"Maths"})

public void testMethod1() {

// test code

}

@Test(priority = 2, groups = {"Maths"})

public void testMethod2() {

// test code

}

### **48. What are Inclusion & Exclusion Groups in TestNG?**

**An inclusion group is a group that is included in test execution. The exclusion group is a group that is excluded from test execution.**

### **49. When do we use "dependsOnGroups" in TestNG?**

TestNG allows us to make a single test rely on a group of tests. We use the **dependsOnGroups** attribute in the TestNG test case file to execute in this manner. The name of the group on which we want this method to rely is the value of this attribute. An illustration of this is provided below:

**import** **org**.testng.annotations.Test;

**public** **class** **MyTests** {

@Test(groups = {"login"})

public void testLogin() {

// test login functionality

}

@Test(groups = {"search"}, dependsOnGroups = {"login"})

public void testSearch() {

// test search functionality

}

@Test(groups = {"checkout"}, dependsOnGroups = {"search"})

public void testCheckout() {

// test checkout functionality

}

}

### ***Assertion related interview Questions for TestNG***

### 50. What do you understand by asserting in TestNG?

[Asserting in TestNG](https://www.lambdatest.com/blog/asserts-in-testng/) is a code element that assists us in determining whether the expected and actual results are equal. To determine whether the test case passed or failed, we use the built-in "Assert" class and many of its methods in TestNG. Furthermore, in TestNG, a test case is considered a "pass" if none of the assert methods throw an exception during execution. The syntax for TestNG assert is as follows:

**import** org.testng.**Assert**;

// **perform** **assertion**

**Assert**.assertEquals(expectedValue, actualValue, message);

### **51. Describe any five common TestNG assertions**

**assertEquals(expectedValue, actualValue, message):** It compares two values and determines if they are equal.

**assertTrue(condition, message):** This assertion helps whether or not the specified condition is true.

**assertFalse(condition, message):** This assertion defines if the specified condition is true or false.

**assertNull(object, message):** This assertion determines whether or not the specified object is null.

**assertNotNull(object, message):** This assertion specifies if the specified object is null.

### **52.** **What are the different types of assert in TestNG?**

* Soft Asserts
* Firm Assertion

### **53. Define soft assert in TestNG and describe how they are different from hard assert**

Soft asserts in TestNG mean that the tests will continue to run even if the assertion throws an exception in the middle of the execution. Furthermore, TestNG does not include Soft asserts by default, so an additional **org.testng.asserts.Softassert** package import is required.

Soft assertions differ from hard assertions in that when a hard assertion fails, the test terminates immediately, and the remaining test steps are skipped. When a soft assertion fails, however, the test continues to run, and any subsequent assertions are also run. Only when the test is completed are the test results, including soft assertion failures, reported, but the test is not terminated.

## Dependency-related interview questions for TestNG

### **54. What is meant by dependency in TestNG?**

Dependencies are used in TestNG to specify the order in which test methods should be executed. A dependency is a relationship between two or more test methods that specifies that one or more tests must be run before another.

@Test(dependsOnMethods = {"testMethod1"})

public void testMethod2(){

// test method code

}

@Test

public void testMethod1(){

// test method code

}

### **55. How do you create dependencies in TestNG?**

In TestNG, we can create dependent tests by specifying the **dependonMethods** parameter on the @Test annotation. The attribute's value is the name of the method on which we want this method to rely. This method is applied as follows:

@Test

public void loginTest() {

// code for login test

}

@Test(dependsOnMethods = {"loginTest"})

public void dashboardTest() {

// code for dashboard test

}

### **56. How do you create dependency through the XML file?**

Through the TestNG XML file, we can also create dependencies between groups. Such dependencies denote a group's reliance on another. Here's an example of how to use the testng.xml file to create a dependency between two test methods:

<test name="MyTest">

<classes>

<**class** name="com.example.MyTestClass">

<methods>

<include name="loginTest" />

<include name="dashboardTest" dependsOnMethods="loginTest" />

</methods>

</class>

</classes>

</test>

### **57. How TestNG allows to declare dependencies?**

TestNG gives two methods to declare dependencies

dependsOnMethods, dependsOnGroups

### **58. How many types of dependencies can you achieve by using TestNG?**

TestNG is a Java testing framework that allows for various types of dependencies between test methods. The following dependencies can be achieved by using TestNG:

|  |  |
| --- | --- |
| **Dependencies** | **Attribute** |
| Method Dependencies | dependsOnMethods |
| Group Dependencies | dependsOnGroups |
| Class Dependencies | dependsOnClasses |
| Soft Dependencies | alwaysRun |
| Priority Dependencies | priority |

## Micillenious Interview questions for TestNG

### 59. What are the priorities in TestNG?

Priorities are used in TestNG to define the order in which test methods should be executed. in the @Test annotation, it is used to assign priorities to test methods. Test methods with lower priority values will be executed first, followed by methods with higher priority values.

### 60. How to set priorities in TestNG?

Priorities for test methods in TestNG can be set using the priority attribute in the @Test annotation. Here's an example:

@Test(priority=1)

public void testMethod1() {

// test code

}

@Test(priority=2)

public void testMethod2() {

// test code

}

@Test(priority=3)

public void testMethod3() {

// test code

}

In this example, testMethod1() is the most important and will be executed first, followed by testMethod2() and, finally, testMethod3(). You can set the execution order of your test methods by assigning any integer value to the priority attribute.

### 61. What is parameterization in TestNG?

[Parameterization in TestNG](https://www.lambdatest.com/blog/parameterization-in-testng-dataprovider-and-testng-xml-examples/), refers to the process of passing parameters to a test method so that it can be run multiple times with different data sets. This feature is especially helpful when running the same test with different input values or test scenarios.

The "@DataProvider" annotation in TestNG can be used to parameterize code. The [DataProvider annotation](https://www.lambdatest.com/blog/how-to-use-dataproviders-in-testng-with-examples/" \t "_blank) specifies a method that provides data to a test method. The data provider method returns a two-dimensional array of objects, with each row representing a set of data used to run the test method.

### 62. What are the optional parameters in TestNG?

Optional parameters behave similarly to the default case in TestNG parameterization. When no other parameters are defined for that test case method, we use the optional parameter. The **@Optional** annotation also declares the optional parameter. We don't define the **@Optional** parameter above the test method definition but rather alongside it.

If defined parameter is not found in your testng.xml file, The test method will receive the default value which is specified inside the @Optional annotation.

Following that, the following code snippet shows how to declare optional parameters in TestNG:

@Parameters("option")

@Test

public void testMethod(@Optional("default") String optionalParam) {

system.out.println(optionlParam);

}

The "**@Optional**" annotation is used in this example to specify a default value for the "optionalParam" parameter. If this parameter option is not supplied from xml then it will print default . it means it will take default. when the test method is run, its value will be "**default.**"

63. What is the significance of "timeout" in TestNG?

The timeout attribute in TestNG is used to specify the maximum amount of time (in milliseconds) that a test method should be allowed to run before being forcefully terminated. This property is useful when a test method becomes stuck in an infinite loop or takes too long to complete, causing the entire test suite to stall. It can be declared at

**suite level:** To impose a time limit on all methods in the suite.

**method level:** To impose a time constraint on a specific method.

**Syntex:**

@**Test**(timeout = 5000)

<test name="MyTest" time-out="5000">

**64. What is meant by invocationCount in TestNG?**

The invocationCount attribute specifies how many times a test method must be executed in a single execution. Hence, if I set the invocationCount to 5, the test function will be invoked five times each time I run the TestNG test case.

@Test (invocationCount = 10)

**65. What is meant by parallel test execution in TestNG?**

[**Parallel test execution in TestNG**](https://www.lambdatest.com/blog/parallel-test-execution-in-testng/)**means running multiple test cases or test suites on multiple threads at the same time.** TestNG is designed to support the parallel execution of tests, which allows testers to run multiple tests concurrently, reducing overall test execution time.

There are two ways to for parallel testing in TestNG:

* Parallel tests at the method level
* Parallel tests at the suite level

**66. On what levels can we apply parallel testing in TestNG?**

Parallel testing is possiblein TestNG at multiple levels:

* **Methods:** This will run all @Test methods in TestNG in parallel.
* **Tests:** This value will be used to run all test cases contained within the <test>tag.
* **Classes:** All of the test cases contained within the XML classes will run in parallel.
* **Instances:** This value will run all test cases concurrently within the same instance.

### 67. How is exception handling done in TestNG?

Exceptions during test execution can be handled in two ways by TestNG:

* The "expectedExceptions" attribute
* The "catch" block method

**Example:**

@Test(expectedExceptions = ArithmeticException.class)

public void testDivideByZero() {

int a = 5;

int b = 0;

int c = a / b;

}

### 68. Can we disable a test in TestNG? If so, explain how?

Yes, you can disable a test in TestNG by annotating the test method with **@Test(enabled=false)**. Once you disable the test, it won't execute when you run the test script again.

Here is an example of it:

@Test(enabled=**true**)

**public** void **enabledTest**() {

// test code here

}

### 69. Why is the reporter class used in TestNG?

During test execution, the Reporter class in TestNG is used to generate additional logs or custom output. It allows developers or testers to log additional information or messages during test execution, which can help with debugging or troubleshooting test failures.

### 70. Define the syntax for generating logs through the reporter class in TestNG

The Reporter.log() method in TestNG can be used to generate logs via the Reporter class.

Reporter.log("TestNG web app testing");

### 71. What are listeners in TestNG?

Listeners are classes in TestNG that can be used to customize and extend TestNG's behavior during test execution. They enable custom actions to be performed before or after specific events, such as the start or end of a test case or suite, the success or failure of a test, or the generation of a test report.

TestNG comes with a set of built-in listeners that you can use right away, or you can create your own custom listeners by implementing the necessary interfaces.

### 72. How to declare listeners in TestNG?

There are two ways to declare listeners in TestNG

* Adding listeners to an XML configuration file:

In the TestNG XML configuration file, add the <listeners> tag inside the <suite> tag to declare listeners. Here's an illustration:

<suite name="My Test Suite">

<listeners>

<listener **class**-name="com.example.MyTestListener"/>

</listeners>

<test name="My Test">

<classes>

<class name="com.example.MyTestClass"/>

</classes>

</test>

</suite>

* Annotating the test class with @Listeners:

Listeners can also be declared by annotating your test class with the @Listeners annotation. Here's an example:

**import** org.testng.annotations.Listeners;

@Listeners(com.example.MyTestListener.class)

**public** **class** **MyTestClass** {

// test methods...

}

### **73. What are the different listeners TestNG provides?**

TestNG includes a number of built-in listeners that you can use to customize and extend your test suite's behavior. Here's a quick rundown of the most common listeners in TestNG:

* IInvokedMethodListener
* IMethodInterceptor
* IAnnotationTransformer
* IAnnotationTransformer2
* IReporter
* ISuiteListener
* IHookable
* ITestListener

### **74. How to Implement TestNG iAnnotationtransformer ?**

To implement the **IAnnotationTransformer** interface in TestNG, you need to create a new Java class that implements the IAnnotationTransformer interface, implement the **transform()** method to modify the **ITestAnnotation** object associated with the test method, and add the listener to your test suite by including it in your TestNG XML configuration file or annotating your test class with the **@Listeners** annotation.

**75. What is the default value for the TestNG Priority?**

The default priority for a test method in TestNG is 0. This means that if a test method is not explicitly assigned a priority using the @Test annotation, it will be assigned a **priority of 0 by default.**

### 76. How to achieve TestNG itestlistener implementation?

Sure, here are the step-by-step instructions for implementing TestNG ITestListener:

* Make a Java class that conforms to the ITestListener interface.
* Override the necessary methods like **onStart(), onTestSuccess(), onTestFailure(), onTestSkipped(),** and **onFinish()**.
* In each of these methods, add the custom actions you want to perform, such as logging, reporting, or modifying test results.
* Make a separate Java file for the listener class.
* Include the tag and specify the full class name of your listener in your TestNG XML configuration file to add the listener class.
* **Alternatively, you can use the @Listeners annotation on your test class and specify the full class name of your listener**.
* When you run your test suite, the listener methods will be executed in response to the test events, allowing you to take custom actions.

### 77. What is an exception test in TestNG?

An exception test in TestNG is a type of test that determines whether or not a method throws a specific exception. When an exception is thrown, it is used to ensure that a method behaves as expected.

To create an exception test in TestNG, use the @Test annotation's **expectedExceptions** attribute. This attribute specifies the type of exception that the method is expected to throw.

### 78. How will you install TestNG in Eclipse?

You can install TestNG in Eclipse by following these steps:

* Navigate to the Help menu in Eclipse.
* From the dropdown menu, select Eclipse Marketplace.
* Enter "TestNG" in the search bar.
* Click the Install button after selecting the TestNG for Eclipse plugin from the search results.
* Accept the license agreement and follow the installation prompts.
* When the installation is finished, restart Eclipse.
* TestNG will be available in Eclipse after installation, and you will be able to create TestNG classes and run TestNG tests within the Eclipse environment.

### 79. How to throw a SKIP Exception in TestNG?

In TestNG, use the following code snippet to throw an SKIP exception:

throw **new** SkipException("Skipping this test **method** **due** **to** **some** **reason**...");

The SkipException class in TestNG is a built-in exception class that can be used to skip a test method during execution. The message passed to the **SkipException c**lass's constructor is displayed in the test results, indicating why the test was skipped.

### 81. What is the time unit we specify in test suites and test cases?

The time unit specified in TestNG test suites and test cases in seconds.

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<**suite** name="MyTestSuite" verbose="1" time-out="180">

<!-- Test suite contents -->

</**suite**>

In this example, we have set the test suite timeout to 180 seconds.

### **82. List out various ways in which TestNG can be invoked?**

TestNG can be launched in a number of different ways.

* With the help of the command line, where you specify the TestNG JAR file and the test suite XML file with the java command.
* Invoke with the integrated development environment (IDE) such as Eclipse, IntelliJ IDEA, or NetBeans.
* You can also use build tools such as Maven, and Gradle
* Use a continuous integration (CI) server such as Jenkins, Travis CI, or CircleCI to run TestNG tests.
* Using a test runner such as the TestNG Eclipse plugin, TestNG Maven plugin, or TestNG Ant task to run TestNG tests.

## TestNG Interview questions for experienced

### 83. How can we create a data-driven framework using TestNG?

You can use TestNG to build a data-driven framework by following these steps:

* Make a TestNG XML file with the test parameters and data provider. A data provider is a method that provides test data to other methods.
* Implement the data provider method to get test data from a data source like a CSV file, Excel sheet, or database.
* Add the data provider attribute to the @Test annotation of the test method to indicate that the data provided by the data provider method will be used by the test method.
* Use the test data to execute the test logic in the test method.

@Test(dataProvider = "testData")

public void testMethod(String firstName, String lastName, int age) {

System.out.println("Name: " + firstName + " " + lastName + ", Age: " + age);

// Test logic using the test data

}

}

### 84. How to exclude a particular test method from a test case execution?

You can use the **enabled** attribute of the **@Test** annotation to exclude a specific test method from a TestNG test case execution. The enabled attribute specifies whether or not a test method should be executed. If the enabled attribute is set to **false**, the test method will be skipped during the execution of the test case.

@Test(enabled = **false**)

**public** void **testMethod2**() {

// Test logic

}

### 85. How to skip a @Test method from execution in TestNG?

The **org.testng.SkipException** class can be used to prevent a @Test method from being executed in TestNG. This class is used to **throw a skip exception,** which instructs TestNG to skip the test method execution. Or we can **do @Test(enabled = false).**

**import** org.testng.SkipException;

**import** org.testng.annotations.Test;

**public** **class** **MyTest** {

@Test

**public** **void** **testMethod2**() {

**throw** **new** SkipException("Skipping testMethod2");

}

}

### 86. How To Run TestNG Using Command Prompt?

C:UsersAdminDesktopLTlearninghubworkspaceTestNGtestingtutoriol

set classpath=C:UsersAdminDesktopLTlearninghubworkspaceTestNGtestingtutoriolin;C:UsersAdminDesktopLTlearninghubworkspaceTestNGtestingtutoriollib\*

java org.testng.TestNG C:UsersAdminDesktopLTlearninghubworkspaceTestNGtestingtutoriol estng.xml

### **87. What is the use of @Test(threadPoolSize=x)?**

In TestNG, **the @Test(threadPoolSize=x)** annotation is used to specify the number of threads to use when running a test method in parallel. The x parameter is an integer value that indicates the thread pool size.

When you use this annotation, TestNG will create a pool of threads and distribute test method instances among them. Each thread will independently and concurrently execute its assigned test method instance with the other threads.

### 88. What is the use of @Test(invocationCount=x)?

In TestNG, the @Test(invocationCount=x) annotation is used to specify how many times a test method should be invoked during the test execution. The x parameter is an integer value that specifies how many times the test method should be run.

When this annotation is used, TestNG will run the test method as many times as specified, each time with a new test instance. This is useful in testing scenarios where the test method must be run multiple times with different input data or conditions.

### **89. What is the thread count in TestNG?**

The thread-count attribute in TestNG is used to specify **the number of threads that will be used to run the test methods.** This attribute is used in the TestNG XML file's <suite> tag.

When the thread-count attribute is set to a value greater than one, TestNG will create multiple threads and distribute the test methods among them for parallel execution.

### **90. What is verbose in TestNG?**

The **verbose** attribute in TestNG is used to limit the amount of information displayed in the console output during test execution.

The verbose attribute accepts integer values ranging from 0 to 10, with 0 being the least verbose and 10 being the most verbose. The default value is 1, which displays basic test execution information.

### **91. How many ways by which can we pass parameter values to test methods?**

We can pass parameter values to test methods in the following ways in TestNG:

* Using @Test annotation parameters
* Making use of the DataProvider method
* Making use of a factory method
* Using an outside data source

### 92. How do you use TestNG to test RESTful web services?

We can use the RestAssured library and write test methods that send HTTP requests and validate the responses to use TestNG to test RESTful web services. To test various scenarios, we can also use TestNG data providers and parameterization.

### 93. How do you use TestNG to test mobile applications?

**Appium, an open-source tool for automating mobile app testing, can be used to test TestNG mobile applications.** To define the desired capabilities of the mobile device, we can create a TestNG test class and write test methods that interact with the mobile app by finding elements and performing actions on them. TestNG annotations can be used to manage the test environment and control the test execution order.

* Install the required software and configure the system variables to create the Appium environment.
* Create a new TestNG test class and specify the capabilities of the mobile device to be tested.
* Create test methods that interact with the mobile app by locating and acting on elements.
* Control the test execution order and manage the test environment with TestNG annotations

### 94. How to implement Data providers in TestNG?

Follow these steps to add data providers to TestNG:

* Create a method that returns an array of two-dimensional objects. Each array row represents a set of test data.

@DataProvider(**name** = "testData")

public Object[][] testData() {

return new Object[][] {

{"Mehul", "password1"},

{"Gadhiya", "password2"},

{"Devya", "password3"}

};

}

* Add the **@DataProvider** annotation to the method and give the data provides a name.
* Add the **dataProvider** attribute to the ,**@Test** annotation in the test method and set it to the name of the data provider.

@Test(dataProvider = "testData")

**public** void **testLogin**(String username, String password) {

// Test code goes here

}

* To receive test data, add parameters to the test method. In the preceding example, we have two parameters, username, and password, which will be set to the data provider's values.
* To perform the test and assert the results, use the test data in the test method.

### **What do you mean by the assertion in Selenium?**

The assertion is used as a verification point. It verifies that the state of the application conforms to what is expected. The types of assertion are "assert", "verify" and "waitFor".

Difference between background and scenario outline?

A Scenario Outline can be used several times to run the same scenario with different input data sets. However the Background section can only be used once per feature file.

**What is feature file in cucumber?**

* A feature file serves as an **entry point** for writing **Cucumber tests**.
* It has **.feature** extension
* Comments: Can be added using the # symbol.
* A feature file can contain **multiple scenarios related to the same feature.**
* Feature: Login
* Scenario: Successful Login
* Given user navigates to the website javatpoint.com
* And there user logs in through Login Window
* by using Username as "USER" and Password as "PASSWORD"
* Then login must be successful

What is scenario in cucumber?

A scenario is a fundamental Gherkin structure. Each scenario begins with the keyword "Scenario:" (or a localized variation of it) at the beginning and ends with a title.

* A scenario represents a specific test case or situation within a feature.
* It starts with the keyword “Scenario:” (or its localized equivalent) and is followed by an optional scenario title.
* Each feature can have one or more scenarios.
* **Every scenario consists of one or more steps.**

How can you pass data between Cucumber steps?

By using Scenario context, a shared data structure that can be accessed by all Cucumber steps in a scenario, allows data to be exchanged across Cucumber steps. Alternatively, data can be transferred between steps using tables or samples of scenario outlines.

Define given when and then keyword ?

* The **Given** step sets up the initial context or preconditions for the test scenario.
* The **When** step represents the action or event that triggers a change in the system. It describes the specific action taken by the user or system.
* The **Then** step defines the expected outcome or result of the action.
* It verifies whether the system behaves as expected.

Tag in cucumber ?

**Scenario Grouping:** Tags help keep related scenarios together (e.g., smoke tests, regression tests).

**Selective Execution:** You can choose to execute only scenarios with specific tags during test runs.

What is a hook in Cucumber?

* **Hooks** are blocks of code that run **before** or **after** each scenario.
* They help you set up necessary conditions before executing a scenario and perform cleanup tasks afterward.
* @Before: This hook runs **before** the execution of any scenario. It’s ideal for setting up prerequisites.
* @After: This hook runs **after** the execution of a scenario. It’s useful for cleanup tasks or reporting.

What is a scenario outline in Cucumber?

 It is used when a scenario needs to be run with various sets of data.

Angle brackets (">") are used to indicate placeholders, which are replaced with actual values during test execution.When I enter "<username>" and "<password>"

### What are the different types of testing that can be performed using Cucumber?

Cucumber is a testing tool that supports various types of testing, such as acceptance testing, functional testing, and regression testing.

* [Acceptance testing](https://www.lambdatest.com/learning-hub/acceptance-testing) includes testing software's performance and compliance with requirements. Cucumber simplifies verifying that the software complies with the required specifications by allowing teams to define acceptance tests in a human-readable language.
* [Functional testing](https://www.lambdatest.com/learning-hub/functional-testing) includes evaluating certain software features or functions to ensure they function as intended. Teams can build and carry out available tests using Cucumber in a language that technical and non-technical stakeholders can easily comprehend.
* [Regression testing](https://www.lambdatest.com/blog/regression-testing-what-is-and-how-to-do-it/) requires retesting the software following changes to ensure that the previously functional features are still operational.

.

### How do you skip a scenario in Cucumber?

Cucumber allows you to skip scenarios by adding **the tag “@ignore” or “@skip.**

How to generate HTML reports in cucumber ?

To create HTML reports, Cucumber comes with a built-in plugin called **HTML Formatter.**

What is a step definition in Cucumber?

A **step definition** is a crucial piece of code that bridges the gap between **Gherkin** steps (expressed in plain language) and the actual implementation in your test automation framework.

Each step in a scenario is matched to a step description by Cucumber before it is executed. Depending on the programming language used to construct the test automation framework, step definitions may be written in Java, Ruby, or JavaScript.

**public class StepDefinitions {**

**@Given("I have {int} cukes in my belly")**

**public void i\_have\_n\_cukes\_in\_my\_belly(int cukes) {**

**System.out.format("Cukes: %d%n%n", cukes);**

**}**

What is cucumber step?

A cucumber step is a specific action or behavior specified in the feature file and put into practice in the step definition file.

|  |  |
| --- | --- |
| **Feature in Cucumber** | **Scenario in Cucumber** |
| A feature in Cucumber is a high-level description of a software need or feature. It often has one or more situations and is defined in a feature file. | A scenario is a particular test case or illustration that confirms a specific behavior or result of the feature. |
| **The "Feature" keyword** is used at the start of a feature in the Gherkin language. | **The "Scenario"** keyword is used at the start of scenarios written in the Gherkin language. |
| Specific test cases can be described by one or more scenarios for a feature. | One or more steps in a scenario can specify the behaviour that will be tested. |

### What is the difference between a scenario and a step in Cucumber?

|  |  |
| --- | --- |
| **Scenario in Cucumber** | **Step in Cucumber** |
| A scenario is a single test case that describes one particular aspect of an application's functionality or behavior. | A step is a distinctive assertion or action that is carried out as a part of a scenario. |
| Consists of one or more steps. | Consists of only one step. |
| Can have a variety of stages with various contexts. | Can have various scenarios with various steps. |
| Used to verify and validate the feature's acceptance criteria. | Used to specify particular test procedures to verify the system's operation. |
| Aids in understanding the system's overall behavior. | Aids in understanding a particular behavior of the system. |

### What is the difference between JUnit and Cucumber?

|  |  |
| --- | --- |
| **JUnit** | **Cucumber** |
| JUnit is a unit testing framework that is generally used to test isolated code modules or units. It offers a collection of annotations and assertions for writing and running tests for Java applications. | Cucumber is a framework for behavior-driven testing that focuses on evaluating a system's overall behavior by specifying scenarios and actions using **Gherkin syntax in natural language.** |
| JUnit uses annotations to specify test methods (@Test, @Before, @After). | Cucumber defines scenarios using the "given-when-then" syntax (Gherkin language) |
| The reading and understanding of tests can be difficult, particularly for non-technical stakeholders. | Scenarios are stated in simple terms so that non-technical stakeholders can understand them. |
| Can integrate with testing tools **like Mockito and Selenium.** | Can integrate with testing tools like Selenium and RestAssured |
| JUnit delivers basic test reporting | Cucumber provides a detailed test reporting in JSON, HTML and other formats |

### What is the difference between RSpec and Cucumber?

|  |  |
| --- | --- |
| **Cucumber** | **RSpec** |
| Cucumber is used for acceptability testing and end-to-end testing | RSpec is mainly used for unit testing. |
| Cucumber's natural language syntax facilitates easier cooperation between technical and non-technical stakeholders. | RSpec's syntax may be more difficult for non-technical stakeholders to understand |
| Cucumber focuses on testing the behavior of an entire system or application | RSpec focuses on testing individual code units. |

### What is the difference between TestNG and Cucumber?

|  |  |
| --- | --- |
| **Cucumber** | **TestNG** |
| Cucumber is used for **acceptability testing** | TestNG is mostly used for unit testing. |
| Cucumber supports a variety of programming languages | TestNG is focused on Java-based testing. |
| Cucumber provides a more natural language approach **using Gherkin** syntax for defining tests | TestNG is more developer-centric and relies on annotations. |
| Cucumber offers a **behavior-driven development** (BDD) strategy that stresses cooperation between developers, testers, and business stakeholders. | TestNG does not have a behavior-driven development (BDD) focus |

### difference between a hook and a step definition in Cucumber?

|  |  |
| --- | --- |
| **Hook in Cucumber** | **Step Definition in Cucumber** |
| **A hook in Cucumber is a section of code that runs before or after a scenario or feature.** | A step definition is a piece of code that specifies a step's behavior in a feature file. |
| Suitable for setup and takedown operations like starting a browser or cutting off a database connection. | Can be used to carry out any automation job, including communicating with UI elements and executing API calls. |
| Using context objects or global variables, data can be shared between scenarios or steps. | Can leverage Cucumber's built-in data sharing features, such as scenario outline tables or background steps, but cannot share data between steps or scenarios. |
| Can be used to conditionally change or skip scenarios depending on specific circumstances. | Can be used to construct conditional logic within a step's implementation, but cannot be used to skip or change scenarios. |

### What is difference between a scenario and a feature file in Cucumber?

The crucial Cucumber components Scenario and Feature both have a specific function. The key variations between them are as follows:

|  |  |
| --- | --- |
| **Feature File** | **Scenario** |
| **A text file called a feature file contains a detailed description of a feature that needs to be tested.** | **A single test case in the Gherkin language is known as a scenario.** |
| It outlines the **software's functioning,** which needs to be tested. | It outlines the specific steps to be taken to test a specific functionality. |
| It can involve a variety of scenarios, each of which tests a distinct function. | It usually comprises of a number of steps that are carried out in a particular order. |
| The scenarios are placed within a context, and they are arranged into logical groups. | To organize and group scenarios, it can be connected to one or more tags. |

### What is the difference between a tag and a data table in Cucumber?

|  |  |
| --- | --- |
| **Tag** | **Data Table** |
| Tag begins with the "@" symbol | Data Table starts with " |
| Tags organize and filter scenarios | Data Table passes numerous parameters to a scenario |
| A scenario may contain several tags. | A scenario can contain only one data table. |
| The step definitions don't use tags. | With step definitions, data tables are used as arguments. |

### difference between a test scenario and a test case in Cucumber?

|  |  |
| --- | --- |
| **Test Scenario** | **Test Case** |
| A test scenario is a specific instance of a feature. | A test case is a set of conditions or variables that a tester will use to verify whether or not a system is functioning properly. |
| Test Scenario is defined using Gherkin syntax and consists of a collection of stages that specify the activities and expected outcomes for specific system behavior. | Test Case is often defined by a QA engineer and contains comprehensive instructions on how to test a particular system feature or behavior. |

|  |
| --- |
| **Test Driven Development** |
| Behavior Driven Development is a development technique which focuses more on a software application’s behavior. | Test Driven Development is a development technique which focuses more on the implementation of a feature of a software application/product. |
| In BDD the participants are Developers, Customer, QAs. | In TDD the participants are developers |

### **What are the various authorization methods provided by Postman?**

Postman provides the following API request authorization options:

* API Key
* Basic auth
* Digest auth
* Hawk Authentication
* Oauth 1.0
* Oauth 2.0
* Bearer Token
* NTLM Authentication

### **What are the different types of API requests supported in Postman?**

The following is a list of the various API requests that Postman supports:

* GET – to retrieve data
* POST- send data to the server
* PUT – It create new resource and replace all existing current repositories .
* PATCH – To update resource and do not update other data .
* COPY-
* DELETE
* HEAD
* OPTIONS
* LINK
* UNLINK
* PURGE
* LOCK
* UNLOCK
* PROPFIND
* VIEW

### **What are the limitations of Postman?**

The following is a list of Postman's main drawbacks:

* Postman cannot process 1000+ API requests.
* It might be challenging to manage collections and requests for large projects.
* For managing the workspace as code, Postman is not appropriate. It is because dynamic API requests would result in a lot of code duplication.

### **. What are Postman methods?**

Postman methods, also known as HTTP methods or HTTP verbs, represent the actions that can be performed on a resource through an API. Some common Postman methods include:

* GET: Retrieves information from a specified resource.
* POST: Submits data to be processed by a specified resource.
* PUT: Updates a specified resource with new data.
* DELETE: Removes a specified resource.
* PATCH: Partially updates a specified resource.

### **Why is the Postman tool used?**

The Postman tool is used for several purposes in API development and testing:

* API Testing: Postman allows you to send requests to APIs and analyze the responses, making it easier to test the functionality, performance, and security of APIs.
* API Documentation: Postman provides features to document APIs, including generating documentation with request examples and descriptions.
* Collaboration: Postman enables team collaboration by allowing you to share requests, collections, and environments with team members.
* Automation: Postman supports the creation and execution of automated tests, making it efficient for continuous integration and delivery (CI/CD) pipelines.
* Mocking: Postman's mocking capabilities enable the creation of simulated API responses, helping developers and testers work independently.
* Monitoring: Postman can be used to monitor API performance and track response times, allowing you to identify potential issues and optimize your APIs.

## What is an API?

**An**[**API**](https://www.postman.com/what-is-an-api/)**, or application programming interface, is a set of protocols that enable different software components to communicate and transfer data**. Developers use private, partner, and public APIs to bridge the gaps between small, discrete chunks of code in order to create applications that are powerful, resilient, secure, and able to meet user needs .

### **What is the use of the collection in Postman?**

**A collection is used to group similar requests in Postman. It systematically arranges the requests into folders.**

To stop the execution of the next request, you can use the following code:

pm.setNextRequest(null)

### **What are the core components of an HTTP request?**

The core components of an HTTP request include:

* Request method (GET, POST, PUT, DELETE, etc.)
* Request URL (Uniform Resource Locator)
* Request headers (e.g., Content-Type, Authorization)
* Request body (optional, used in POST or PUT requests)

### **40. Why does Postman accept Base64 encoding only?**

Postman accepts Base64 encoding as it is a widely-used format for encoding binary data into a text string. Base64 encoding is used to handle binary data safely within HTTP requests and responses.