Module-4

Automation Core Testing (Load Runner Up and Selenium IDE)

1. **Which components have you used in Load Runner?**

- **Virtual User Generator (VuGen):** VuGen is used for recording user actions on the application to create scripts for performance testing. It supports multiple scripting languages like C, Java, and VBScript.
- **Controller:** The Controller component is used to design and manage the execution of load testing scenarios. It allows testers to configure the number of virtual users, define scenarios, set up load distribution, and monitor test execution.
- **Load Generators:** Load generators are machines or virtual machines responsible for executing the scripts generated by VuGen. They simulate multiple virtual users to generate load on the application under test.
- **Agent Process:** The Agent process runs on load generators to manage and execute Vuser scripts. It communicates with the Controller to receive instructions and report back the execution status and performance metrics.
- **Analysis:** LoadRunner's Analysis component is used for analysing and interpreting the results of performance tests. It provides various graphs, charts, and reports to visualize performance metrics and identify bottlenecks.
- **Monitoring Tools:** LoadRunner integrates with various monitoring tools to capture performance metrics from the application, web servers, databases, and other infrastructure components during test execution. Examples include SiteScope for server monitoring and Performance Centre for enterprise-level monitoring.
- **Protocols:** LoadRunner supports a wide range of protocols for recording and scripting different types of applications, including web-based applications (HTTP/HTML), web services (SOAP, REST), database protocols (JDBC, ODBC), messaging protocols (MQ), and more.

These components work together to facilitate the entire performance testing process, from script creation to test execution, monitoring, analysis, and reporting.

2. **How can you set the number of Vusers in Load Runner?**

In LoadRunner, the number of Virtual Users (Vusers) can be set through the LoadRunner Controller. Here's how you can do it:

- **Open LoadRunner Controller:** Launch the LoadRunner Controller application.
- **Create or Open a Scenario:** You can either create a new scenario or open an existing one.
- **Define Vuser Groups:** In LoadRunner, virtual users are organized into groups called Vuser groups. Each Vuser group represents a set of users executing a specific set of actions. You can create multiple Vuser groups to simulate different user behaviors.
- **Set the Number of Vusers:** Within each Vuser group, you can specify the number of virtual users to simulate. This can be done by entering the desired number in the "Number of Vusers" field or by using the sliders provided.
- **Distribute Vusers:** If you have multiple load generators, you can distribute the Vusers across these generators. LoadRunner allows you to specify how many Vusers should run on each load generator.
- **Configure Ramp-Up:** Ramp-up refers to the gradual increase in the number of Vusers over time. You can configure ramp-up settings to simulate realistic user behaviour, such as a gradual increase in user load during peak hours.
- **Run the Scenario:** Once you've configured the number of Vusers and other settings, you can start the scenario execution. The Controller will distribute the specified number of Vusers across the load generators and begin simulating user activity on the application under test.
- **Monitor Execution:** During scenario execution, you can monitor various performance metrics such as response times, throughput, and server resource utilization to assess the application's performance under load.

By adjusting the number of Vusers and other parameters in the LoadRunner Controller, you can simulate different levels of user load and analyse how the application behaves under various conditions.

Top of Form

3. What is Correlation?

Correlation, in the context of LoadRunner or performance testing, refers to the process of capturing and replacing dynamic values in a recorded script with parameters. Dynamic values are those that change with each user session or interaction, such as session IDs, authentication tokens, timestamps, or any other unique identifiers generated by the server.

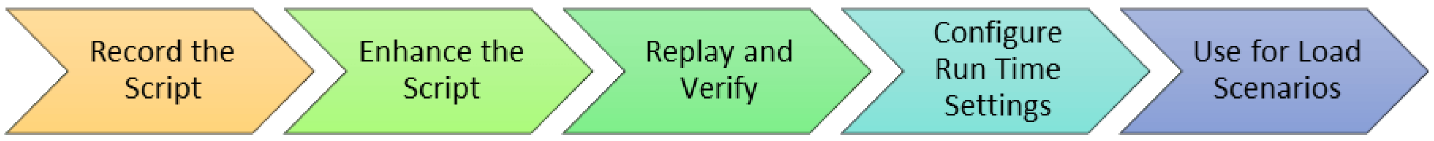
Here's why correlation is necessary and how it works:

- **Dynamic Values:** Modern web applications often use dynamic values to maintain state or security. For example, a session ID might be generated for each user session, or an authentication token might be issued upon login.
- **Recording Scripts:** When you record a script using a performance testing tool like LoadRunner, it captures all the interactions between the user and the application, including these dynamic values.
- **Replay:** When you replay the recorded script, the dynamic values captured during recording may no longer be valid because they were specific to the session that was recorded. If you use the same values during replay, the server may reject the requests, leading to script failure.
- **Correlation:** To address this issue, correlation involves identifying these dynamic values in the recorded script and replacing them with parameters. Parameters are placeholders that are dynamically populated with values during script execution. This process ensures that each virtual user gets a unique set of values, mimicking real user behaviour.
- **Identifying Dynamic Values:** Correlation requires identifying the dynamic values within the script. This can be done manually by examining the recorded script for patterns or automatically using built-in tools provided by performance testing tools like LoadRunner.
- **Extracting and Parameterizing:** Once identified, the dynamic values are extracted from the server responses and replaced with parameters in the script. LoadRunner provides mechanisms to automatically correlate commonly used dynamic values, and testers can also create custom correlation rules as needed.
- **Replay and Validation:** After correlation, the script is replayed to ensure that it runs successfully without errors. Testers may need to iterate the correlation process until all dynamic values are properly handled, and the script executes as expected.

Overall, correlation is a critical step in script development for performance testing, ensuring that scripts accurately simulate real user behaviour and produce reliable test results.

4. What is the process for developing a Vuser Script?

- **The script development process in VUGen**



- **Record the Script:** Usually, this is the first step of scripting where every user

action is recorded into a script.

- **Replay and Verify:** Once the script is recorded, reply the script to ensure its working right. Verify any impact through application frontend or database.
- **Enhance the Script:** Once recording has been verified, enhance script by adding checkpoints, validating data, adding transactions and rendezvous points.
- **Replay and Verify:** As earlier, re-play your script and verify that everything is working as intended.
- **Configure Runtime Settings:** Configure and control pacing duration, think time variation, proxy settings and whether you wish to ignore any external resources.
- **Use for Load Scenarios:** Formulate load scenarios based on test objectives. Use load distribution and geo-wide agents to make real like scenarios.

5. How Load Runner interacts with the application?

LoadRunner interacts with applications primarily through a process called performance testing. LoadRunner is a performance testing tool that simulates user activity on software applications, allowing testers to measure and analyse system performance under various conditions.

Here's how LoadRunner typically interacts with an application:

- **Scripting:** Testers create scripts using LoadRunner's scripting language, usually by recording user interactions or manually coding the scenarios. These scripts simulate user actions like logging in, browsing pages, filling forms, etc.
- **Parameterization:** LoadRunner allows for parameterization, where variables can be used to represent dynamic data such as user names, passwords, or input data. This enables realistic simulation of user behaviour.
- **Scenario Design:** Testers design scenarios that represent different types of user loads and behaviours. For example, a scenario might simulate a specific number of users logging in simultaneously or performing specific actions at different rates.
- **Controller Setup:** LoadRunner's Controller component orchestrates the execution of scenarios. Testers configure the desired number of virtual users, ramp-up rates, and other parameters to mimic real-world usage patterns.

- **Execution:** The Controller distributes the load across multiple load generators (machines or virtual machines running LoadRunner's Virtual User Generator or Vuser scripts). Each load generator simulates multiple virtual users executing the scripts against the application under test.
- **Monitoring:** During test execution, LoadRunner collects performance metrics such as response times, throughput, CPU and memory usage, database performance, etc., from the application servers, web servers, and other infrastructure components.
- **Analysis:** After the test completes, testers analyse the collected data using LoadRunner's Analysis component. They can identify performance bottlenecks, pinpoint areas for optimization, and evaluate the application's scalability, reliability, and responsiveness under different loads.
- **Reporting:** LoadRunner generates comprehensive reports summarizing the test results, including graphs, charts, and statistics, to help stakeholders understand the application's performance characteristics and make informed decisions.

Overall, LoadRunner provides a robust framework for assessing an application's performance under realistic conditions, helping organizations ensure that their software meets performance objectives and user expectations.

6. How many VUsers are required for load testing?

Determining the number of Virtual Users (VUsers) required for load testing depends on several factors, including the objectives of the test, the nature of the application, the expected user load in production, and the available infrastructure. Here are some considerations:

- **Business Requirements:** Start by understanding the business requirements and objectives of the load test. What performance metrics are you aiming to measure or validate? This could include response times, throughput, concurrency, resource utilization, etc.
- **Expected User Load:** Determine the expected user load on the application in production. This can be based on historical data, expected user growth, or business projections. You'll want to simulate a load that represents realistic usage patterns to identify performance bottlenecks.
- **Scalability Goals:** If the application is expected to scale to handle a large number of users, you may want to conduct scalability testing to determine its limits. This may involve gradually increasing the number of VUsers until performance degrades or reaches a threshold.
- **Infrastructure Capacity:** Consider the capacity of your load testing infrastructure, including the number of load generators available and their capabilities. Ensure that

your infrastructure can support the desired number of VUsers without resource contention or bottlenecks.

- **Test Scenarios:** Define different test scenarios representing various user activities and usage patterns. Each scenario may require a different number of VUsers. For example, you may have scenarios for login, browsing, searching, purchasing, etc.
- **Ramp-Up Strategy:** Decide on a ramp-up strategy for gradually increasing the number of VUsers during the test. This helps simulate realistic user behaviour and allows you to observe how the system performs under increasing load.
- **Monitoring and Analysis:** During the test, closely monitor performance metrics to identify bottlenecks and areas for optimization. You may need to adjust the number of VUsers or test scenarios based on the observed performance characteristics.
- **Iterations:** Load testing is often an iterative process. You may need to conduct multiple test iterations, adjusting the number of VUsers and other parameters based on the results of previous tests.

Ultimately, there is no one-size-fits-all answer to how many VUsers are required for load testing. It's essential to tailor the test approach to the specific requirements and characteristics of the application being tested and to iteratively refine the testing strategy based on results and observations.

7. What is the relationship between Response Time and Throughput?

Response time and throughput are two key performance metrics used in load testing and performance monitoring. While they are related, they measure different aspects of system performance.

- **Response Time:**

Response time, also known as latency, is the time it takes for the system to respond to a user request. It includes the time from when the request is sent until the user receives a complete response.

Response time is typically measured in milliseconds or seconds.

A lower response time indicates better system performance and responsiveness from the user's perspective.

- **Throughput:**

Throughput measures the number of transactions processed by the system within a given period, usually per second.

It represents the system's processing capacity and how many requests it can handle simultaneously.

Throughput is typically measured in transactions per second (TPS) or requests per second (RPS).

Higher throughput indicates better system scalability and the ability to handle more concurrent users or transactions.

- **Relationship:**

While response time and throughput are related, they are not directly proportional. In some cases, as throughput increases, response time may also increase, and vice versa.

A system with high throughput may still have high response times if it's struggling to process a large number of concurrent requests.

Conversely, a system with low throughput may have low response times if it's not under heavy load.

The relationship between response time and throughput is influenced by various factors, including system resources, concurrency, network latency, application design, and workload characteristics.

8. **To test the Performance testing on “Tops Technologies website” :- https://www.saucedemo.com/**

- **to Record all top level menu**
- **to Record minimum 10 Vuser on this website**
- **save all (Script,Design,Graph)**

```
Action()
{

        web_url("gts1c3.der",
                "URL=http://pki.goog/repo/certs/gts1c3.der",
                "Resource=1",
                "RecContentType=application/pkix-cert",
                "Referer=",
                "Snapshot=t5.inf",
                LAST);

        web_url("gtsr1.der",
                "URL=http://pki.goog/repo/certs/gtsr1.der",
                "Resource=1",
			"RecContentType=application/pkix-cert",
			"Referer=",
			"Snapshot=t6.inf",
			LAST);

		web_url("gts1c3.der_2",
			"URL=http://pki.goog/repo/certs/gts1c3.der",
			"Resource=1",
			"RecContentType=application/pkix-cert",
			"Referer=",
			"Snapshot=t7.inf",
			LAST);

		web_url("gtsr1.der_2",
			"URL=http://pki.goog/repo/certs/gtsr1.der",
			"Resource=1",
			"RecContentType=application/pkix-cert",
			"Referer=",
			"Snapshot=t8.inf",
			LAST);

		web_url("RapidSSLTLSRSACAG1.crt",
			"URL=http://cacerts.rapidssl.com/RapidSSLTLSRSACAG1.crt",
			"Resource=1",
			"RecContentType=application/pkix-cert",
			"Referer=",
			"Snapshot=t9.inf",
			LAST);

		return 0;
	}
```

9. **create a normal script of above website with correlate using hp default website.**

```
Action()
{

	web_url("index.htm",
		"URL=http://127.0.0.1:1080/WebTours/index.htm",
		"Resource=0",
		"Referer=",
		"Snapshot=t1.inf",
		"Mode=HTML",
		LAST);

	web_url("header.html",
		"URL=http://127.0.0.1:1080/WebTours/header.html",
		"Resource=0",
		"Referer=http://127.0.0.1:1080/WebTours/index.htm",
		"Snapshot=t2.inf",
		"Mode=HTML",
		LAST);

	web_url("welcome.pl",
		"URL=http://127.0.0.1:1080/cgi-bin/welcome.pl?signOff=true",
		"Resource=0",
		"RecContentType=text/html",
		"Referer=http://127.0.0.1:1080/WebTours/index.htm",
		"Snapshot=t3.inf",
		"Mode=HTML",
		EXTRARES,
		"Url=http://pki.goog/repo/certs/gts1c3.der", "Referer=", ENDITEM,
		"Url=http://pki.goog/repo/certs/gtsr1.der", "Referer=", ENDITEM,
		LAST);

	lr_save_string(lr_decrypt("6620c77998a0b4f6"), "PasswordParameter");

	web_submit_data("login.pl",
		"Action=http://127.0.0.1:1080/cgi-bin/login.pl",
		"Method=POST",
		"RecContentType=text/html",
		"Referer=http://127.0.0.1:1080/cgi-bin/nav.pl?in=home",
		"Snapshot=t4.inf",
		"Mode=HTML",
		ITEMDATA,
		"Name=userSession",
"Value=138794.996921831HVDQczHpzftVzzzHtciDfpzAfQcf", ENDITEM,
		"Name=username", "Value=jojo", ENDITEM,
		"Name=password", "Value={PasswordParameter}", ENDITEM,
		"Name=login.x", "Value=47", ENDITEM,
		"Name=login.y", "Value=8", ENDITEM,
		"Name=JSFormSubmit", "Value=off", ENDITEM,
		LAST);
```

```
        return 0;
    }
```

10. What is Automation Testing?

Automation testing is a software testing technique that involves using automated tools and scripts to execute test cases and verify the behaviour and performance of software applications. Instead of manual intervention, automation testing relies on pre-written scripts to perform repetitive and complex test scenarios, thereby increasing testing efficiency, repeatability, and accuracy.

11. Which Are The Browsers Supported By Selenium Ide?

Selenium IDE (Integrated Development Environment) is primarily a Firefox extension. It's designed as a browser extension for Mozilla Firefox and is tightly integrated with the Firefox browser.

However, Selenium IDE has been evolving, and there have been efforts to support other browsers as well.

As of my last update in January 2022, Selenium IDE (Integrated Development Environment) is primarily a Firefox extension. It's designed as a browser extension for Mozilla Firefox and is tightly integrated with the Firefox browser.

However, Selenium IDE has been evolving, and there have been efforts to support other browsers as well. Here's the browser support status as of my last update:

1. **Mozilla Firefox:** Selenium IDE is primarily developed for Mozilla Firefox and offers the most comprehensive support for this browser.
2. **Google Chrome:** Selenium IDE also provides experimental support for Google Chrome. There is a separate Chrome extension available, but the functionality may be limited compared to the Firefox version, and it may still be in development or testing phases.
3. **Microsoft Edge:** Selenium IDE may also have experimental support for Microsoft Edge, either through a dedicated Edge extension or compatibility with Chrome extensions (since Edge is based on Chromium).
4. **Other Browsers:** Support for other browsers may vary, and it's recommended to check the latest documentation or announcements from the Selenium IDE project for updates on browser support.

12. What are the benefits of Automation Testing?

Automation testing offers numerous benefits that contribute to improving the efficiency, effectiveness, and quality of software development processes. Here are some of the key benefits:

- **Faster Time-to-Market:** Automation testing enables faster execution of test cases, allowing for quicker identification and resolution of bugs. This accelerates the software development lifecycle and helps deliver products to market more rapidly.
- **Increased Testing Coverage:** Automated tests can cover a wide range of test scenarios, including regression testing, integration testing, and performance testing. This ensures comprehensive test coverage, including edge cases and negative scenarios, which may be impractical to test manually.
- **Repeatability and Consistency:** Automated tests produce consistent and reproducible results, reducing the risk of human error and ensuring consistent test coverage across different test runs and environments.
- **Cost Savings:** While automation testing requires an initial investment in setup and development, it ultimately saves time and resources by reducing the need for manual testing efforts. Automated tests can be run repeatedly without additional costs, making them more cost-effective in the long run.
- **Improved Accuracy:** Automated tests execute test cases with precision and accuracy, minimizing the likelihood of false positives or false negatives. This improves the reliability of test results and helps identify genuine issues more effectively.
- **Early Detection of Defects:** Automation testing enables early detection of defects in the software development lifecycle, allowing teams to address issues promptly and prevent them from escalating into more significant problems later in the development process.
- **Support for Continuous Integration/Continuous Deployment (CI/CD):** Automation testing is well-suited for integration into CI/CD pipelines, enabling automated testing of code changes with each build or deployment. This facilitates rapid feedback loops and ensures that new features or changes are thoroughly tested before being released to production.
- **Scalability:** Automated tests can handle large test suites and complex scenarios, making them suitable for testing applications with evolving requirements and increasing complexity.
- **Enhanced Developer Productivity:** Automation testing frees up developers from repetitive and time-consuming testing tasks, allowing them to focus on higher-value activities such as feature development, code optimization, and innovation.

- **Improved Software Quality:** By identifying and addressing defects early in the development process, automation testing contributes to improving the overall quality, reliability, and stability of software applications, leading to higher customer satisfaction and loyalty.

Overall, automation testing is a valuable practice that offers numerous benefits for software development teams, helping them deliver high-quality products more efficiently and effectively.

13. What are the advantages of Selenium?

Selenium is a widely used open-source tool for automating web browsers. It offers several advantages that make it a preferred choice for automated testing of web applications:

- **Cross-Browser Compatibility:** Selenium supports multiple web browsers, including Google Chrome, Mozilla Firefox, Microsoft Edge, Safari, and others. This cross-browser compatibility allows testers to validate the consistency of web applications across different browsers and platforms.
- **Platform Independence:** Selenium is a platform-independent tool, meaning it can run on various operating systems such as Windows, macOS, Linux, and Unix. This flexibility allows testers to develop and execute tests on their preferred operating system without any compatibility issues.
- **Support for Multiple Programming Languages:** Selenium supports multiple programming languages, including Java, Python, C#, Ruby, and JavaScript. Testers can write automated test scripts using their preferred programming language, making Selenium accessible to a wide range of developers and testers.
- **Integration with Testing Frameworks:** Selenium can be easily integrated with popular testing frameworks such as JUnit, TestNG, NUnit, and Pytest. This integration allows testers to leverage advanced testing features such as test reporting, parallel execution, data-driven testing, and more.
- **Rich Set of Features:** Selenium offers a rich set of features for automating web browser interactions, including form filling, clicking buttons, navigating links, handling alerts and pop-ups, capturing screenshots, and validating page content. These features enable testers to simulate real user interactions and thoroughly test web applications.
- **Community Support:** Selenium has a large and active community of developers and testers who contribute to its development, share knowledge, and provide support through forums, blogs, and online communities. This community-driven approach ensures timely updates, bug fixes, and enhancements to the Selenium framework.
- **Extensibility:** Selenium is highly extensible, allowing users to customize and extend its functionality through third-party plugins, libraries, and frameworks. Testers can

enhance Selenium's capabilities by integrating it with additional tools for tasks such as test data generation, reporting, and continuous integration.

- **Open-Source:** Selenium is an open-source tool distributed under the Apache License 2.0, making it freely available for use, modification, and distribution. Organizations can leverage Selenium without incurring licensing costs, making it a cost-effective solution for test automation.
- **Browser Interaction Simulation:** Selenium provides a set of APIs for simulating user interactions with web browsers, enabling testers to automate complex testing scenarios such as filling forms, uploading files, handling cookies, and executing JavaScript actions.
- **Support for Headless Testing:** Selenium supports headless browser testing, allowing testers to run automated tests without launching a graphical user interface. Headless testing is useful for running tests in environments without a GUI, such as continuous integration servers or containerized environments.

Overall, Selenium offers a powerful and flexible platform for automating web browser testing, helping organizations improve the efficiency, reliability, and quality of their web applications.

14. Why testers should opt for Selenium and not QTP?

Choosing between Selenium and QTP (now known as UFT - Unified Functional Testing) depends on various factors, including the project requirements, budget, skillset of the testing team, and the specific needs of the organization. Here are some reasons why testers might opt for Selenium over QTP/UFT:

- **Open Source vs. Commercial Tool:** Selenium is an open-source tool, meaning it's freely available for use, modification, and distribution. On the other hand, QTP/UFT is a commercial tool, which requires purchasing licenses, leading to higher upfront costs and ongoing maintenance fees.
- **Platform Independence:** Selenium is platform-independent and supports multiple operating systems, browsers, and programming languages. It can be used on Windows, macOS, Linux, and Unix systems, and it supports various browsers like Chrome, Firefox, Edge, Safari, etc. QTP/UFT, while versatile, may have limitations in terms of platform support and may require additional configurations or plugins for cross-platform testing.
- **Programming Language Support:** Selenium supports multiple programming languages, including Java, Python, C#, Ruby, and JavaScript. Testers can write test scripts in their preferred language, leveraging their existing skills and knowledge. QTP/UFT primarily uses VBScript, which may not be as widely used or preferred by all testers.

- **Community Support and Flexibility:** Selenium has a large and active community of developers and testers who contribute to its development, share knowledge, and provide support. This community-driven approach ensures timely updates, bug fixes, and enhancements to the Selenium framework. QTP/UFT, while supported by its vendor, may have fewer resources and community contributions.
- **Integration with Testing Frameworks:** Selenium can be easily integrated with popular testing frameworks such as TestNG, JUnit, NUnit, and Pytest. These frameworks provide advanced testing features such as test reporting, parallel execution, data-driven testing, and more. QTP/UFT also supports integration with testing frameworks, but Selenium's flexibility and extensibility may offer more options and customization capabilities.
- **Web Application Testing Focus:** Selenium is primarily designed for web application testing and excels in automating web browser interactions. It provides robust features for simulating user interactions, validating page content, handling alerts and pop-ups, and executing JavaScript actions. QTP/UFT, while capable of testing web applications, also supports testing of desktop, mobile, and API-based applications, which may be unnecessary for organizations focused solely on web application testing.
- **Cost Considerations:** Selenium's open-source nature eliminates licensing costs, making it a cost-effective option for organizations with budget constraints. QTP/UFT, being a commercial tool, requires purchasing licenses for each user, which can be expensive, especially for larger testing teams.

Ultimately, the choice between Selenium and QTP/UFT depends on the specific needs and priorities of the organization, as well as factors such as budget, expertise, and project requirements. While Selenium offers many advantages, QTP/UFT may still be a suitable option for organizations that prioritize commercial support, comprehensive testing capabilities, and integration with other Micro Focus products.