DUMPSBOSS.

Certified Professional Selenium Tester (CPST)

GAQM CPST-001

Version Demo

Total Demo Questions: 12

Total Premium Questions: 125

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

QUESTION NO: 1

The following is used to run the exported tests in any browser and on any platform?

A. Selenium 2

B. Selenium Core

C. Selenium Remote Control

D. None of the above

ANSWER: C

Explanation:

Selenium Remote Control is correct in the historical Selenium architecture assumed by this question. Selenium IDE could record interactions and export the resulting tests into programming-language formats. Selenium Remote Control then provided the server-and-client-library arrangement used to execute those exported tests against browsers. The RC server acted as an intermediary between the test code and the browser, enabling test execution across supported browser types and operating-system platforms rather than limiting execution to the environment in which a test was recorded.

This capability was a central reason for using Selenium Remote Control with exported Selenium IDE tests: the exported script could be integrated into a language-specific test framework and run remotely or on different machines and browser configurations. Selenium's legacy documentation describes Selenium RC as the earlier remote-control approach and its role in driving browsers through a server component. Although Selenium RC has since been superseded by Selenium WebDriver, it is the terminology and component that matches the question's reference to exported tests. See the Selenium 1 (legacy) documentation and Selenium's project overview documentation for the evolution from RC to WebDriver.

QUESTION NO: 2

After a user submits a form, a loading overlay temporarily covers the Save button. Select two synchronization practices that allow the test to continue only when a user can safely click Save.

A. Wait explicitly until the loading overlay is invisible.

B. Wait explicitly until the Save button is clickable.

C. Use `Thread.sleep()` for a fixed interval after every submission.

D. Set a long implicit wait and click the button immediately.

ANSWER: A B

Explanation:

An explicit wait for the loading overlay to become invisible addresses the condition preventing interaction, and an explicit wait for the Save button to become clickable verifies that the target is visible and enabled for interaction. A fixed sleep merely delays the test for an arbitrary period and can be either insufficient or unnecessarily slow. An implicit wait is intended for element-location attempts and does not specifically wait for an overlay to disappear or for clickability.

QUESTION NO: 3

Which of the following commands is not a type of assertion in Selenium IDE?

A. WaitFor

B. Assert

C. Wait

D. Verify

ANSWER: C

Explanation:

`Wait` is correct because Selenium IDE's traditional assertion-related command families are identified by the `assert`, `verify`, and `waitFor` prefixes. These prefixes are combined with a condition or subject, such as checking element text, presence, visibility, or another expected page state. An assertion evaluates an expected condition; an `assert` command stops the test when the condition fails, whereas a `verify` command records the failure and allows the test to continue. The `waitFor` family repeatedly checks a condition until it becomes true or a timeout is reached, which makes it useful for synchronizing a test with dynamic page behavior. By contrast, `Wait` alone is not one of those assertion-command prefixes or a complete traditional Selenium IDE assertion type. Selenium IDE's command reference documents the available commands and their behavior, while Selenium's general documentation explains the importance of explicit waiting for application states before continuing automated interactions. See the Selenium IDE command reference and the Selenium WebDriver waits documentation.

QUESTION NO: 4

How many types of reports can be generated for Java in Selenium?

A. No count and are still emerging

B. 4

C. 2

D. Not there

ANSWER: B

Explanation:

4 is the correct answer in the CPST Java/Selenium training context. Selenium automates browser interactions and supplies test-execution information to the Java test framework and reporting integrations; it does not limit a project to one fixed report file. The course material treats Java Selenium reporting as four report types/categories, reflecting the commonly taught use of framework-produced reports and dedicated reporting solutions. These reports turn test outcomes, execution times, failures, and supporting diagnostic details into information that testers and stakeholders can review after a run. In practical Java projects, reporting is normally produced by the test runner or a reporting library used alongside Selenium. For example, TestNG documents its built-in reporting facilities, including listener-based reporting and generated output, while Selenium's Java documentation describes Selenium WebDriver support for Java-based automated tests. Therefore, the expected numerical response is four, rather than a claim that reporting is unavailable or uncounted. See the TestNG reporters documentation and the Selenium WebDriver documentation for the respective reporting and Java automation context.

QUESTION NO: 5

A TestNG suite contains `createCustomer()` and `placeOrder()`, where an order is meaningful only after a customer has been created. Select two correct statements about using TestNG method dependencies.

A. `@Test(dependsOnMethods = "createCustomer")` can declare the prerequisite for `placeOrder()`.

B. `placeOrder()` is normally skipped if `createCustomer()` fails.

C. Setting a lower `priority` on `createCustomer()` provides the same failure dependency.

D. A method dependency allows `placeOrder()` to run successfully without customer test data.

ANSWER: A B

Explanation:

Using `dependsOnMethods` expresses the required execution relationship, and TestNG normally skips the dependent test when its prerequisite fails. This prevents a misleading order failure caused solely by missing prerequisite data. A dependency does not make the tests independent, and a priority setting alone only influences scheduling; it does not enforce the prerequisite-success rule.

QUESTION NO: 6

What are the different navigation methods of WebDriver?

A. back()

B. moveto()

C. forward()

D. goto()

E. refresh()

ANSWER: A C E

Explanation:

`back()`, `forward()`, and `refresh()` are WebDriver browser-navigation methods exposed through the `Navigation` interface, normally accessed with `driver.navigate()`. They model the navigation controls available in a browser: `driver.navigate().back()` returns the browser to the preceding entry in its history, `driver.navigate().forward()` moves to the next history entry after a back operation, and `driver.navigate().refresh()` reloads the current page. These commands are especially useful in end-to-end Selenium tests that must verify behavior across page transitions, restore a previous application state via browser history, or confirm that a page continues to work after reloading. Selenium's Java API documents these operations directly on `WebDriver.Navigation`, including the corresponding `back`, `forward`, and `refresh` methods. The Selenium documentation also presents browser navigation as a distinct WebDriver capability, separate from element interaction and mouse movement. See the WebDriver.Navigation Java API and Selenium's browser navigation documentation.

QUESTION NO: 7

A TestNG project labels a small set of tests as smoke tests and must execute only that set from its suite configuration. Select two correct ways to define and use the `smoke` group.

A. Assign the group with `@Test(groups = {"smoke"})`.

B. Include `smoke` in the suite XML group-run configuration.

C. Set every smoke test to `priority = 1`.

D. Rename each smoke-test class to `SmokeTest`.

ANSWER: A B

Explanation:

A TestNG method can be assigned to a group through the `groups` attribute of `@Test`. The suite XML can then include that group through its `<groups>` and `<run>` configuration, causing only methods in the named group to be selected. Method priority changes execution order but does not create group membership, and a group is not defined by renaming a Java class.

QUESTION NO: 8

What are the types of exceptions thrown in web driver?

A. ElementNotSelectableException

B. NoElementFoundException

C. ElementNotVisibleException

D. NoSuchElementException

E. StaleElementReferenceException

ANSWER: A C D E

Explanation:

`ElementNotSelectableException`, `ElementNotVisibleException`, `NoSuchElementException`, and `StaleElementReferenceException` are recognized Selenium WebDriver exceptions in the Java bindings and are appropriate answers to this broad question. `NoSuchElementException` represents a failed element-location attempt. `StaleElementReferenceException` represents a reference to a DOM element that is no longer attached to the current page document, commonly after page refreshes or dynamic DOM updates. `ElementNotSelectableException` is part of Selenium's exception hierarchy for an element that cannot be selected. `ElementNotVisibleException` was a Selenium exception used when an element was present but not visible for the required interaction. In newer Selenium releases it is deprecated in favor of more specific interaction-related exceptions, but it remains a valid legacy WebDriver exception type and is therefore correctly included when the question does not specify a Selenium version. Selenium's Java API documents the exception hierarchy and the current interaction exception model; see the Selenium NoSuchElementException API documentation and the Selenium StaleElementReferenceException API documentation.

QUESTION NO: 9

A TestNG test must execute the same search scenario with several search terms. Select two correct steps for implementing this with a DataProvider.

A. Return the search data sets from a method annotated with `@DataProvider`.

B. Reference that provider with `@Test(dataProvider = "providerName")`.

C. Declare every search term only with the `@Parameters` annotation.

D. Create a separate `testng.xml` file for each search term.

ANSWER: A B

Explanation:

A method annotated with `@DataProvider` supplies one or more data sets, commonly as an `Object[][]`, and the test refers to it through the `dataProvider` attribute of `@Test`. TestNG invokes the test once for each supplied data set. A DataProvider does not require a `testng.xml` parameter for every value, and `@Parameters` is a separate mechanism for values supplied through suite configuration.

QUESTION NO: 10

How do you start Selenium RC?

A. Start Selenium RC Server

B. java -jar selenium-server-standalone-.jar

C. Start RC

D. None of the above

ANSWER: B

Explanation:

The command `java -jar selenium-server-standalone-<version-number>.jar` starts the Selenium Server process from a standalone server JAR. Selenium RC uses a server component that must be running before an RC client can send browser-automation commands to it. Running the JAR with Java launches that server and makes it available to receive requests, traditionally on its default server port unless configuration arguments specify otherwise. The placeholder `<version-number>` must be replaced with the version included in the downloaded JAR filename; for example, a historical distribution might use a filename such as `selenium-server-standalone-2.x.x.jar`. This is the expected command-line mechanism reflected in Selenium RC-era training material. Selenium RC is now a legacy technology, and modern Selenium projects normally use Selenium WebDriver rather than creating new RC-based automation. Nevertheless, when operating an existing RC setup, launching the standalone server JAR through the Java runtime is the required startup action. Selenium's legacy documentation describes the Selenium RC server architecture and startup model at Selenium RC legacy documentation. Current Selenium server command-line usage is documented at Selenium documentation.

QUESTION NO: 11

The following is used to add the assert or verify to the test:

A. Selenium IDE

B. Context Menu

C. Slider bar

D. Base URL tab

ANSWER: B

Explanation:

Context Menu is correct because Selenium IDE supports adding validation commands based on a selected page element through its contextual, right-click menu. When an element is identified in the browser, the context menu can offer Selenium IDE actions that generate assertion or verification steps using an appropriate locator and the element's current property, such as its text, value, presence, or visibility. This provides a convenient way to extend a recorded test with an expected result without manually composing every command and locator.

Assertions and verifications are central to a Selenium IDE test because they check whether the application is in the expected state at a particular point in the execution. Selenium IDE's command reference distinguishes assertion-style commands, which enforce an expected condition, from related verification behavior used to check outcomes during a test. The context-menu workflow is especially useful during test authoring: the tester identifies the relevant UI element, chooses the desired assertion or verification action, and Selenium IDE inserts the corresponding command into the test case. See the Selenium IDE getting-started documentation and the Selenium IDE command reference for command behavior and supported validation commands.

QUESTION NO: 12

A navigation menu displays its submenu only while the pointer is over the parent menu item. Which Selenium Java sequence is most appropriate for opening the submenu before clicking an item inside it?

A. Use `new Actions(driver).moveToElement(menu).click(submenuItem).perform();`

B. Use `menu.click();` and assume that the submenu becomes visible.

C. Use `driver.navigate().to(submenuUrl);` to simulate the hover.

D. Use `driver.switchTo().frame(menu);` before clicking the submenu.

ANSWER: A

Explanation:

The `Actions` API is designed for composite user interactions such as moving the pointer to an element and clicking a revealed submenu item. Building the action sequence and calling `perform()` executes it against the browser. Calling `click()` on the parent may activate a link rather than create the required hover state, while JavaScript navigation does not verify the browser interaction expected from a user.