1) **How you used WinRunner in your project?**
   a) Yes, I have been using WinRunner for creating automated scripts for GUI, functional and regression testing of the AUT.
2) **Explain WinRunner testing process?**
   a) WinRunner testing process involves six main stages
      i. **Create GUI Map** File so that WinRunner can recognize the GUI objects in the application being tested
      ii. **Create test** scripts by recording, programming, or a combination of both. While recording tests, insert checkpoints where you want to check the response of the application being tested.
      iii. **Debug Test:** run tests in Debug mode to make sure they run smoothly
      iv. **Run Tests:** run tests in Verify mode to test your application.
      v. **View Results:** determines the success or failure of the tests.
      vi. **Report Defects:** If a test run fails due to a defect in the application being tested, you can report information about the defect directly from the Test Results window.
3) **What is contained in the GUI map?**
   a) WinRunner stores information it learns about a window or object in a GUI Map. When WinRunner runs a test, it uses the GUI map to locate objects. It reads an object's description in the GUI map and then looks for an object with the same properties in the application being tested. Each of these objects in the GUI Map file will be having a logical name and a physical description.
   b) There are 2 types of GUI Map files.
      i. **Global GUI Map file**: a single GUI Map file for the entire application
      ii. **GUI Map File per Test:** WinRunner automatically creates a GUI Map file for each test created.
4) **How does WinRunner recognize objects on the application?**
   a) WinRunner uses the GUI Map file to recognize objects on the application. When WinRunner runs a test, it uses the GUI map to locate objects. It reads an object's description in the GUI map and then looks for an object with the same properties in the application being tested.
5) **Have you created test scripts and what is contained in the test scripts?**
   a) Yes I have created test scripts. It contains the statement in Mercury Interactive's Test Script Language (TSL). These statements appear as a test script in a test window. You can then enhance your recorded test script, either by typing in additional TSL functions and programming elements or by using WinRunner's visual programming tool, the Function Generator.
6) **How does WinRunner evaluates test results?**
   a) Following each test run, WinRunner displays the results in a report. The report details all the major events that occurred during the run, such as checkpoints, error messages, system messages, or user messages. If mismatches are detected at checkpoints during the test run, you can view the expected results and the actual results from the Test Results window.

7) **Have you performed debugging of the scripts?**
   a) Yes, I have performed debugging of scripts. We can debug the script by executing the script in the debug mode. We can also debug script using the Step, Step Into, Step out functionalities provided by the WinRunner.

8) **How do you run your test scripts?**
   a) We run tests in Verify mode to test your application. Each time WinRunner encounters a checkpoint in the test script, it compares the current data of the application being tested to the expected data captured earlier. If any mismatches are found, WinRunner captures them as actual results.

9) **How do you analyze results and report the defects?**
   a) Following each test run, WinRunner displays the results in a report. The report details all the major events that occurred during the run, such as checkpoints, error messages, system messages, or user messages. If mismatches are detected at checkpoints during the test run, you can view the expected results and the actual results from the Test Results window. If a test run fails due to a defect in the application being tested, you can report information about the defect directly from the Test Results window. This information is sent via e-mail to the quality assurance manager, who tracks the defect until it is fixed.

10) **What is the use of Test Director software?**
   a) TestDirector is Mercury Interactive's software test management tool. It helps quality assurance personnel plan and organize the testing process. With TestDirector you can create a database of manual and automated tests, build test cycles, run tests, and report and track defects. You can also create reports and graphs to help review the progress of planning tests, running tests, and tracking defects before a software release.

11) **How you integrated your automated scripts from TestDirector?**
   a) When you work with WinRunner, you can choose to save your tests directly to your TestDirector database or while creating a test case in the TestDirector we can specify whether the script in automated or manual. And if it is automated script then TestDirector will build a skeleton for the script that can be later modified into one which could be used to test the AUT.

12) **What are the different modes of recording?**
   a) There are two type of recording in WinRunner.
      i. **Context Sensitive recording** records the operations you perform on your application by identifying Graphical User Interface (GUI) objects.
      ii. **Analog recording** records keyboard input, mouse clicks, and the precise x- and y-coordinates traveled by the mouse pointer across the screen.

13) **What is the purpose of loading WinRunner Add-Ins?**
   a) Add-Ins are used in WinRunner to load functions specific to the particular add-in to the memory. While creating a script only those functions in the

add-in selected will be listed in the function generator and while executing the script only those functions in the loaded add-in will be executed else WinRunner will give an error message saying it does not recognize the function.

14) What are the reasons that WinRunner fails to identify an object on the GUI?

a) WinRunner fails to identify an object in a GUI due to various reasons.
   i. The object is not a standard windows object.
   ii. If the browser used is not compatible with the WinRunner version, GUI Map Editor will not be able to learn any of the objects displayed in the browser window.

15) What do you mean by the logical name of the object?

a) An object's logical name is determined by its class. In most cases, the logical name is the label that appears on an object.

16) If the object does not have a name then what will be the logical name?

a) If the object does not have a name then the logical name could be the attached text.

17) What is the different between GUI map and GUI map files?

a) The GUI map is actually the sum of one or more GUI map files. There are two modes for organizing GUI map files.
   i. **Global GUI Map file**: a single GUI Map file for the entire application
   ii. **GUI Map File per Test:** WinRunner automatically creates a GUI Map file for each test created.

b) GUI Map file is a file which contains the windows and the objects learned by the WinRunner with its logical name and their physical description.

18) How do you view the contents of the GUI map?

a) GUI Map editor displays the content of a GUI Map. We can invoke GUI Map Editor from the Tools Menu in WinRunner. The GUI Map Editor displays the various GUI Map files created and the windows and objects learned in to them with their logical name and physical description.

19) When you create GUI map do you record all the objects of specific objects?

a) If we are learning a window then WinRunner automatically learns all the objects in the window else we will we identifying those object, which are to be learned in a window, since we will be working with only those objects while creating scripts.

20) What is the purpose of set_window command?

a) Set_Window command sets the focus to the specified window. We use this command to set the focus to the required window before executing tests on a particular window.

Syntax: *set_window(<logical name>, time);*
The logical name is the logical name of the window and time is the time the execution has to wait till it gets the given window into focus.

21) How do you load GUI map?

a) We can load a GUI Map by using the GUI_load command.

Syntax: *GUI_load(<file_name>);*

22) What is the disadvantage of loading the GUI maps through start up scripts?

a) If we are using a single GUI Map file for the entire AUT then the memory used by the GUI Map may be much high.

b) If there is any change in the object being learned then WinRunner will not be able to recognize the object, as it is not in the GUI Map file loaded in the memory. So we will have to learn the object again and update the GUI File and reload it.

23) How do you unload the GUI map?

a) We can use GUI_close to unload a specific GUI Map file or else we call use GUI_close_all command to unload all the GUI Map files loaded in the memory.

Syntax: *GUI_close(<file_name>); or GUI_close_all;*

24) What actually happens when you load GUI map?

a) When we load a GUI Map file, the information about the windows and the objects with their logical names and physical description are loaded into memory. So when the WinRunner executes a script on a particular window, it can identify the objects using this information loaded in the memory.

25) What is the purpose of the temp GUI map file?

a) While recording a script, WinRunner learns objects and windows by itself. This is actually stored into the temporary GUI Map file. We can specify whether we have to load this temporary GUI Map file should be loaded each time in the General Options.

26) What is the extension of gui map file?

a) The extension for a GUI Map file is ".gui".

27) How do you find an object in an GUI map.

a) The GUI Map Editor is been provided with a **Find** and **Show** Buttons.

   i. To find a particular object in the GUI Map file in the application, select the object and click the **Show** window. This blinks the selected object.

   ii. To find a particular object in a GUI Map file click the **Find** button, which gives the option to select the object. When the object is selected, if the object has been learned to the GUI Map file it will be focused in the GUI Map file.

28) What different actions are performed by find and show button?

a) To find a particular object in the GUI Map file in the application, select the object and click the **Show** window. This blinks the selected object.

b) To find a particular object in a GUI Map file click the **Find** button, which gives the option to select the object. When the object is selected, if the object has been learned to the GUI Map file it will be focused in the GUI Map file.

29) How do you identify which files are loaded in the GUI map?

a) The GUI Map Editor has a drop down "**GUI File**" displaying all the GUI Map files loaded into the memory.

30) How do you modify the logical name or the physical description of the objects in GUI map?

a) You can modify the logical name or the physical description of an object in a GUI map file using the GUI Map Editor.

31) When do you feel you need to modify the logical name?

a) Changing the logical name of an object is useful when the assigned logical name is not sufficiently descriptive or is too long.

32) When it is appropriate to change physical description?

a) Changing the physical description is necessary when the property value of an object changes.

33) How WinRunner handles varying window labels?

a) We can handle varying window labels using **regular expressions.** WinRunner uses two "hidden" properties in order to use regular expression in an object's physical description. These properties are **regexp_label** and **regexp_MSW_class**.

   i. The **regexp_label** property is used for windows only. It operates "behind the scenes" to insert a regular expression into a window's label description.

   ii. The **regexp_MSW_class** property inserts a regular expression into an object's **MSW_class**. It is obligatory for all types of windows and for the object class object.

[

34) What is the purpose of regexp_label property and regexp_MSW_class property?

a) The **regexp_label** property is used for windows only. It operates "behind the scenes" to insert a regular expression into a window's label description.

b) The **regexp_MSW_class** property inserts a regular expression into an object's **MSW_class**. It is obligatory for all types of windows and for the object class object.

35) How do you suppress a regular expression?

a) We can suppress the regular expression of a window by replacing the **regexp_label** property with **label** property.

36) How do you copy and move objects between different GUI map files?

a) We can copy and move objects between different GUI Map files using the GUI Map Editor. The steps to be followed are:

   i. Choose **Tools > GUI Map Editor** to open the GUI Map Editor.

   ii. Choose **View > GUI Files**.

   iii. Click **Expand** in the **GUI Map Editor**. The dialog box expands to display two GUI map files simultaneously.

   iv. View a different GUI map file on each side of the dialog box by clicking the file names in the GUI File lists.

   v. In one file, select the objects you want to copy or move. Use the **Shift key and/or Control key to select multiple objects**. To select all objects in a GUI map file, choose **Edit > Select All**.

   vi. Click **Copy or Move**.

   vii. To restore the GUI Map Editor to its original size, click **Collapse**.

37) **How do you select multiple objects during merging the files?**

a) Use the **Shift key and/or Control key to select multiple objects**. To select all objects in a GUI map file, choose **Edit > Select All**.

38) **How do you clear a GUI map files?**

a) We can clear a GUI Map file using the "**Clear All**" option in the GUI Map Editor.

39) **How do you filter the objects in the GUI map?**

a) GUI Map Editor has a Filter option. This provides for filtering with 3 different types of options.

   i. Logical name displays only objects with the specified logical name.

   ii. Physical description displays only objects matching the specified physical description. Use any substring belonging to the physical description.

   iii. Class displays only objects of the specified class, such as all the push buttons.

40) **How do you configure GUI map?**

a) When WinRunner learns the description of a GUI object, it does not learn all its properties. Instead, it learns the minimum number of properties to provide a unique identification of the object.

b) Many applications also contain custom GUI objects. A custom object is any object not belonging to one of the standard classes used by WinRunner. These objects are therefore assigned to the generic "object" class. When WinRunner records an operation on a custom object, it generates **obj_mouse_** statements in the test script.

c) If a custom object is similar to a standard object, you can map it to one of the standard classes. You can also configure the properties WinRunner uses to identify a custom object during Context Sensitive testing. The mapping and the configuration you set are valid only for the current WinRunner session. To make the mapping and the configuration permanent, you must add configuration statements to your startup test script.

41) **What is the purpose of GUI map configuration?**

a) GUI Map configuration is used to map a custom object to a standard object.

42) **How do you make the configuration and mappings permanent?**

a) The mapping and the configuration you set are valid only for the current WinRunner session. To make the mapping and the configuration permanent, you must add configuration statements to your startup test script.

43) **What is the purpose of GUI spy?**

a) Using the GUI Spy, you can view the properties of any GUI object on your desktop. You use the Spy pointer to point to an object, and the GUI Spy displays the properties and their values in the GUI Spy dialog box. You can choose to view all the properties of an object, or only the selected set of properties that WinRunner learns.

44) **What is the purpose of obligatory and optional properties of the objects?**

  a) For each class, WinRunner learns a set of default properties. Each default property is classified "**obligatory**" or "**optional**".

    i. An **obligatory** property is always learned (if it exists).

    ii. An **optional** property is used only if the obligatory properties do not provide unique identification of an object. These optional properties are stored in a list. WinRunner selects the minimum number of properties from this list that are necessary to identify the object. It begins with the first property in the list, and continues, if necessary, to add properties to the description until it obtains unique identification for the object.

45) **When the optional properties are learned?**

  a) An optional property is used only if the obligatory properties do not provide unique identification of an object.

46) **What is the purpose of location indicator and index indicator in GUI map configuration?**

  a) In cases where the obligatory and optional properties do not uniquely identify an object, WinRunner uses a selector to differentiate between them. Two types of selectors are available:

    i. A **location selector** uses the spatial position of objects.

      1. The location selector uses the spatial order of objects within the window, from the top left to the bottom right corners, to differentiate among objects with the same description.

    ii. An **index selector** uses a unique number to identify the object in a window.

      1. The index selector uses numbers assigned at the time of creation of objects to identify the object in a window. Use this selector if the location of objects with the same description may change within a window.

47) **How do you handle custom objects?**

  a) A custom object is any GUI object not belonging to one of the standard classes used by WinRunner. WinRunner learns such objects under the generic "object" class. WinRunner records operations on custom objects using **obj_mouse_** statements.

  b) If a custom object is similar to a standard object, you can map it to one of the standard classes. You can also configure the properties WinRunner uses to identify a custom object during Context Sensitive testing.

48) **What is the name of custom class in WinRunner and what methods it applies on the custom objects?**

  a) WinRunner learns custom class objects under the generic "object" class. WinRunner records operations on custom objects using **obj_** statements.

49) **In a situation when obligatory and optional both the properties cannot uniquely identify an object what method WinRunner applies?**

  a) In cases where the obligatory and optional properties do not uniquely identify an object, WinRunner uses a selector to differentiate between them. Two types of selectors are available:

    i. A **location selector** uses the spatial position of objects.

    ii. An **index selector** uses a unique number to identify the object in a window.

50) What is the purpose of different record methods 1) Record 2) Pass up 3) As Object 4) Ignore.

a) **Record** instructs WinRunner to record all operations performed on a GUI object. This is the default record method for all classes. (The only exception is the static class (static text), for which the default is Pass Up.)

b) **Pass Up** instructs WinRunner to record an operation performed on this class as an operation performed on the element containing the object. Usually this element is a window, and the operation is recorded as **win_mouse_click.**

c) **As Object** instructs WinRunner to record all operations performed on a GUI object as though its class were "object" class.

d) **Ignore** instructs WinRunner to disregard all operations performed on the class.

51) How do you find out which is the start up file in WinRunner?

a) The test script name in the Startup Test box in the Environment tab in the General Options dialog box is the start up file in WinRunner.

52) What are the virtual objects and how do you learn them?

a) Applications may contain bitmaps that look and behave like GUI objects. WinRunner records operations on these bitmaps using win_mouse_click statements. By defining a bitmap as a virtual object, you can instruct WinRunner to treat it like a GUI object such as a push button, when you record and run tests.

b) Using the Virtual Object wizard, you can assign a bitmap to a standard object class, define the coordinates of that object, and assign it a logical name.

To define a virtual object using the Virtual Object wizard:

i. Choose Tools > Virtual Object Wizard. The Virtual Object wizard opens. Click Next.

ii. In the Class list, select a class for the new virtual object. If rows that are displayed in the window. For a table class, select the number of visible rows and columns. Click Next.

iii. Click Mark Object. Use the crosshairs pointer to select the area of the virtual object. You can use the arrow keys to make precise adjustments to the area you define with the crosshairs. Press Enter or click the right mouse button to display the virtual object's coordinates in the wizard. If the object marked is visible on the screen, you can click the Highlight button to view it. Click Next.

iv. Assign a logical name to the virtual object. This is the name that appears in the test script when you record on the virtual object. If the object contains text that WinRunner can read, the wizard suggests using this text for the logical name. Otherwise, WinRunner suggests **virtual_object**, **virtual_push_button**, **virtual_list**, etc.

v. You can accept the wizard's suggestion or type in a different name. WinRunner checks that there are no other objects in the GUI map with the same name before confirming your choice. Click Next.

53) How you created you test scripts 1) by recording or 2) programming?

a) Programming. I have done complete programming only, absolutely no recording.

54) What are the two modes of recording?

a) There are 2 modes of recording in WinRunner

  i. **Context Sensitive recording** records the operations you perform on your application by identifying Graphical User Interface (GUI) objects.

  ii. **Analog recording** records keyboard input, mouse clicks, and the precise x- and y-coordinates traveled by the mouse pointer across the screen.

55) What is a checkpoint and what are different types of checkpoints?

a) Checkpoints allow you to compare the current behavior of the application being tested to its behavior in an earlier version.

You can add four types of checkpoints to your test scripts:

  i. **GUI checkpoints** verify information about GUI objects. For example, you can check that a button is enabled or see which item is selected in a list.

  ii. **Bitmap checkpoints** take a "snapshot" of a window or area of your application and compare this to an image captured in an earlier version.

  iii. **Text checkpoints** read text in GUI objects and in bitmaps and enable you to verify their contents.

  iv. **Database checkpoints** check the contents and the number of rows and columns of a result set, which is based on a query you create on your database.

56) What are data driven tests?

a) When you test your application, you may want to check how it performs the same operations with multiple sets of data. You can create a data-driven test with a loop that runs ten times: each time the loop runs, it is driven by a different set of data. In order for WinRunner to use data to drive the test, you must link the data to the test script which it drives. This is called parameterizing your test. The data is stored in a data table. You can perform these operations manually, or you can use the DataDriver Wizard to parameterize your test and store the data in a data table.

57) What are the synchronization points?

a) Synchronization points enable you to solve anticipated timing problems between the test and your application. For example, if you create a test that opens a database application, you can add a synchronization point that causes the test to wait until the database records are loaded on the screen.

b) For Analog testing, you can also use a synchronization point to ensure that WinRunner repositions a window at a specific location. When you run a test, the mouse cursor travels along exact coordinates. Repositioning the window enables the mouse pointer to make contact with the correct elements in the window.

58) What is parameterizing?

a) In order for WinRunner to use data to drive the test, you must link the data to the test script which it drives. This is called parameterizing your test. The data is stored in a data table.

59) How do you maintain the document information of the test scripts?

a) Before creating a test, you can document information about the test in the General and Description tabs of the Test Properties dialog box. You can enter the name of the test author, the type of functionality tested, a detailed description of the test, and a reference to the relevant functional specifications document.

60) What do you verify with the GUI checkpoint for single property and what command it generates, explain syntax?

a) You can check a single property of a GUI object. For example, you can check whether a button is enabled or disabled or whether an item in a list is selected. To create a GUI checkpoint for a property value, use the Check Property dialog box to add one of the following functions to the test script:

   i. button_check_info
   ii. scroll_check_info
   iii. edit_check_info
   iv. static_check_info
   v. list_check_info
   vi. win_check_info
   vii. obj_check_info

Syntax: *button_check_info (button, property, property_value);*
edit_check_info (edit, property, property_value);

61) What do you verify with the GUI checkpoint for object/window and what command it generates, explain syntax?

a) You can create a GUI checkpoint to check a single object in the application being tested. You can either check the object with its default properties or you can specify which properties to check.

b) Creating a GUI Checkpoint using the Default Checks

   i. You can create a GUI checkpoint that performs a default check on the property recommended by WinRunner. For example, if you create a GUI checkpoint that checks a push button, the default check verifies that the push button is enabled.

   ii. To create a GUI checkpoint using default checks:

      1. Choose Create > GUI Checkpoint > For Object/Window, or click the GUI Checkpoint for Object/Window button on the User toolbar. If you are recording in Analog mode, press the CHECK GUI FOR OBJECT/WINDOW softkey in order to avoid extraneous mouse movements. Note that you can press the CHECK GUI FOR OBJECT/WINDOW softkey in Context Sensitive mode as well. The WinRunner window is minimized, the mouse pointer becomes a pointing hand, and a help window opens on the screen.
      2. Click an object.

3. WinRunner captures the current value of the property of the GUI object being checked and stores it in the test's expected results folder. The WinRunner window is restored and a GUI checkpoint is inserted in the test script as an obj_check_gui statement

Syntax: *win_check_gui (window, checklist, expected_results_file, time);*

c) **Creating a GUI Checkpoint by Specifying which Properties to Check**

d) You can specify which properties to check for an object. For example, if you create a checkpoint that checks a push button, you can choose to verify that it is in focus, instead of enabled.

e) **To create a GUI checkpoint by specifying which properties to check:**

- **i.** Choose Create > GUI Checkpoint > For Object/Window, or click the GUI Checkpoint for Object/Window button on the User toolbar. If you are recording in Analog mode, press the CHECK GUI FOR OBJECT/WINDOW softkey in order to avoid extraneous mouse movements. Note that you can press the CHECK GUI FOR OBJECT/WINDOW softkey in Context Sensitive mode as well. The WinRunner window is minimized, the mouse pointer becomes a pointing hand, and a help window opens on the screen.
- **ii.** Double-click the object or window. The Check GUI dialog box opens.
- **iii.** Click an object name in the Objects pane. The Properties pane lists all the properties for the selected object.
- **iv.** Select the properties you want to check.
  1. To edit the expected value of a property, first select it. Next, either click the Edit Expected Value button, or double-click the value in the Expected Value column to edit it.
  2. To add a check in which you specify arguments, first select the property for which you want to specify arguments. Next, either click the Specify Arguments button, or double-click in the Arguments column. Note that if an ellipsis (three dots) appears in the Arguments column, then you must specify arguments for a check on this property. (You do not need to specify arguments if a default argument is specified.) When checking standard objects, you only specify arguments for certain properties of edit and static text objects. You also specify arguments for checks on certain properties of nonstandard objects.
  3. To change the viewing options for the properties of an object, use the Show Properties buttons.
  4. Click OK to close the Check GUI dialog box. WinRunner captures the GUI information and stores it in the test's expected results folder. The WinRunner window is restored and a GUI checkpoint is inserted in the test script as an obj_check_gui or a win_check_gui statement.

Syntax: *win_check_gui (window, checklist, expected_results_file, time);*
obj_check_gui (object, checklist, expected results file, time);

62) **What do you verify with the GUI checkpoint for multiple objects and what command it generates, explain syntax?**

a) **To create a GUI checkpoint for two or more objects:**

   i. Choose Create > GUI Checkpoint > For Multiple Objects or click the GUI Checkpoint for Multiple Objects button on the User toolbar. If you are recording in Analog mode, press the CHECK GUI FOR MULTIPLE OBJECTS softkey in order to avoid extraneous mouse movements. The Create GUI Checkpoint dialog box opens.

   ii. Click the Add button. The mouse pointer becomes a pointing hand and a help window opens.

   iii. To add an object, click it once. If you click a window title bar or menu bar, a help window prompts you to check all the objects in the window.

   iv. The pointing hand remains active. You can continue to choose objects by repeating step 3 above for each object you want to check.

   v. Click the right mouse button to stop the selection process and to restore the mouse pointer to its original shape. The Create GUI Checkpoint dialog box reopens.

   vi. The Objects pane contains the name of the window and objects included in the GUI checkpoint. To specify which objects to check, click an object name in the Objects pane. The Properties pane lists all the properties of the object. The default properties are selected.

      1. To edit the expected value of a property, first select it. Next, either click the Edit Expected Value button, or double-click the value in the Expected Value column to edit it.
      2. To add a check in which you specify arguments, first select the property for which you want to specify arguments. Next, either click the Specify Arguments button, or double-click in the Arguments column. Note that if an ellipsis appears in the Arguments column, then you must specify arguments for a check on this property. (You do not need to specify arguments if a default argument is specified.) When checking standard objects, you only specify arguments for certain properties of edit and static text objects. You also specify arguments for checks on certain properties of nonstandard objects.
      3. To change the viewing options for the properties of an object, use the Show Properties buttons.

   vii. To save the checklist and close the Create GUI Checkpoint dialog box, click OK. WinRunner captures the current property values of the selected GUI objects and stores it in the expected results folder. A win_check_gui statement is inserted in the test script.

Syntax: *win_check_gui (window, checklist, expected_results_file, time);*
obj_check_gui (object, checklist, expected results file, time);

63) **What information is contained in the checklist file and in which file expected results are stored?**
   a) The checklist file contains information about the objects and the properties of the object we are verifying.
   b) The **gui*.chk** file contains the expected results which is stored in the **exp** folder

64) **What do you verify with the bitmap check point for object/window and what command it generates, explain syntax?**
   a) You can check an object, a window, or an area of a screen in your application as a bitmap. While creating a test, you indicate what you want to check. WinRunner captures the specified bitmap, stores it in the expected results folder (exp) of the test, and inserts a checkpoint in the test script. When you run the test, WinRunner compares the bitmap currently displayed in the application being tested with the expected bitmap stored earlier. In the event of a mismatch, WinRunner captures the current actual bitmap and generates a difference bitmap. By comparing the three bitmaps (expected, actual, and difference), you can identify the nature of the discrepancy.
   b) When working in Context Sensitive mode, you can capture a bitmap of a window, object, or of a specified area of a screen. WinRunner inserts a checkpoint in the test script in the form of either a win_check_bitmap or obj_check_bitmap statement.
   c) Note that when you record a test in Analog mode, you should press the CHECK BITMAP OF WINDOW softkey or the CHECK BITMAP OF SCREEN AREA softkey to create a bitmap checkpoint. This prevents WinRunner from recording extraneous mouse movements. If you are programming a test, you can also use the Analog function check_window to check a bitmap.

   d) **To capture a window or object as a bitmap:**

      i. Choose Create > Bitmap Checkpoint > For Object/Window or click the Bitmap Checkpoint for Object/Window button on the User toolbar. Alternatively, if you are recording in Analog mode, press the CHECK BITMAP OF OBJECT/WINDOW softkey. The WinRunner window is minimized, the mouse pointer becomes a pointing hand, and a help window opens.
      ii. Point to the object or window and click it. WinRunner captures the bitmap and generates a win_check_bitmap or obj_check_bitmap statement in the script. The TSL statement generated for a window bitmap has the following syntax:
      win_check_bitmap (object, bitmap, time);
      iii. For an object bitmap, the syntax is:
      obj_check_bitmap (object, bitmap, time);

      iv. For example, when you click the title bar of the main window of the Flight Reservation application, the resulting statement might be:
      win_check_bitmap ("Flight Reservation", "Img2", 1);

v. However, if you click the Date of Flight box in the same window, the statement might be:

obj_check_bitmap ("Date of Flight:", "Img1", 1);

Syntax: *obj_check_bitmap (object, bitmap, time [, x, y, width, height]);*

65) **What do you verify with the bitmap checkpoint for screen area and what command it generates, explain syntax?**

a) You can define any rectangular area of the screen and capture it as a bitmap for comparison. The area can be any size: it can be part of a single window, or it can intersect several windows. The rectangle is identified by the coordinates of its upper left and lower right corners, relative to the upper left corner of the window in which the area is located. If the area intersects several windows or is part of a window with no title (for example, a popup window), its coordinates are relative to the entire screen (the root window).

b) **To capture an area of the screen as a bitmap:**

i. Choose Create > Bitmap Checkpoint > For Screen Area or click the Bitmap Checkpoint for Screen Area button. Alternatively, if you are recording in Analog mode, press the CHECK BITMAP OF SCREEN AREA softkey. The WinRunner window is minimized, the mouse pointer becomes a crosshairs pointer, and a help window opens.

ii. Mark the area to be captured: press the left mouse button and drag the mouse pointer until a rectangle encloses the area; then release the mouse button.

iii. Press the right mouse button to complete the operation. WinRunner captures the area and generates a win_check_bitmap statement in your script.

iv. The win_check_bitmap statement for an area of the screen has the following syntax:

win_check_bitmap (window, bitmap, time, x, y, width, height);

66) **What do you verify with the database checkpoint default and what command it generates, explain syntax?**

a) By adding runtime database record checkpoints you can compare the information in your application during a test run with the corresponding record in your database. By adding standard database checkpoints to your test scripts, you can check the contents of databases in different versions of your application.

b) When you create database checkpoints, you define a query on your database, and your database checkpoint checks the values contained in the result set. The result set is set of values retrieved from the results of the query.

c) You can create runtime database record checkpoints in order to compare the values displayed in your application during the test run with the corresponding values in the database. If the comparison does not meet the success criteria you

d) specify for the checkpoint, the checkpoint fails. You can define a successful runtime database record checkpoint as one where one or more matching records were found, exactly one matching record was found, or where no matching records are found.
e) You can create standard database checkpoints to compare the current values of the properties of the result set during the test run to the expected values captured during recording or otherwise set before the test run. If the expected results and the current results do not match, the database checkpoint fails. Standard database checkpoints are useful when the expected results can be established before the test run.

   Syntax: *db_check(<checklist_file>, <expected_restult>);*

f) You can add a runtime database record checkpoint to your test in order to compare information that appears in your application during a test run with the current value(s) in the corresponding record(s) in your database. You add runtime database record checkpoints by running the Runtime Record Checkpoint wizard. When you are finished, the wizard inserts the appropriate **db_record_check** statement into your script.

   Syntax:
   db_record_check(ChecklistFileName,SuccessConditions,RecordNumber);

   ChecklistFileName A file created by WinRunner and saved in the test's checklist folder. The file contains information about the data to be captured during the test run and its corresponding field in the database. The file is created based on the information entered in the Runtime Record Verification wizard.
   SuccessConditions Contains one of the following values:
   1. DVR_ONE_OR_MORE_MATCH - The checkpoint passes if one or more matching database records are found.
   2. DVR_ONE_MATCH - The checkpoint passes if exactly one matching database record is found.
   3. DVR_NO_MATCH - The checkpoint passes if no matching database records are found.

   RecordNumber An out parameter returning the number of records in the database.

67) How do you handle dynamically changing area of the window in the bitmap checkpoints?

a) The difference between bitmaps option in the Run Tab of the general options defines the minimum number of pixels that constitute a bitmap mismatch

68) What do you verify with the database check point custom and what command it generates, explain syntax?

a) When you create a custom check on a database, you create a standard database checkpoint in which you can specify which properties to check on a result set.
b) You can create a custom check on a database in order to:

i. check the contents of part or the entire result set
ii. edit the expected results of the contents of the result set
iii. count the rows in the result set
iv. count the columns in the result set

c) You can create a custom check on a database using ODBC, Microsoft Query or Data Junction.

69) What do you verify with the sync point for object/window property and what command it generates, explain syntax?

a) Synchronization compensates for inconsistencies in the performance of your application during a test run. By inserting a synchronization point in your test script, you can instruct WinRunner to suspend the test run and wait for a cue before continuing the test.

b) You can a synchronization point that instructs WinRunner to wait for a specified object or window to appear. For example, you can tell WinRunner to wait for a window to open before performing an operation within that window, or you may want WinRunner to wait for an object to appear in order to perform an operation on that object.

c) You use the obj_exists function to create an object synchronization point, and you use the win_exists function to create a window synchronization point. These functions have the following syntax:

Syntax:

obj_exists (object [, time]);
win_exists (window [, time]);

70) What do you verify with the sync point for object/window bitmap and what command it generates, explain syntax?

a) You can create a bitmap synchronization point that waits for the bitmap of an object or a window to appear in the application being tested.

b) During a test run, WinRunner suspends test execution until the specified bitmap is redrawn, and then compares the current bitmap with the expected one captured earlier. If the bitmaps match, then WinRunner continues the test.

Syntax:

obj_wait_bitmap (object, image, time);
win_wait_bitmap (window, image, time);

71) What do you verify with the sync point for screen area and what command it generates, explain syntax?

a) For screen area verification we actually capture the screen area into a bitmap and verify the application screen area with the bitmap file during execution

Syntax: *obj_wait_bitmap(object, image, time, x, y, width, height);*

72) How do you edit checklist file and when do you need to edit the checklist file?

a) WinRunner has an edit checklist file option under the create menu. Select the "Edit GUI Checklist" to modify GUI checklist file and "Edit Database Checklist" to edit database checklist file. This brings up a dialog box that gives you option to select the checklist file to modify. There is also an option to select the scope of the checklist file, whether it is Test specific or a shared one. Select the checklist file, click OK which opens up the window to edit the properties of the objects.

73) **How do you edit the expected value of an object?**

a) We can modify the expected value of the object by executing the script in the Update mode. We can also manually edit the gui*.chk file which contains the expected values which come under the exp folder to change the values.

74) **How do you modify the expected results of a GUI checkpoint?**

a) We can modify the expected results of a GUI checkpoint be running the script containing the checkpoint in the update mode.

75) **How do you handle ActiveX and Visual basic objects?**

a) WinRunner provides with add-ins for ActiveX and Visual basic objects. When loading WinRunner, select those add-ins and these add-ins provide with a set of functions to work on ActiveX and VB objects.

76) **How do you create ODBC query?**

a) We can create ODBC query using the database checkpoint wizard. It provides with option to create an SQL file that uses an ODBC DSN to connect to the database. The SQL File will contain the connection string and the SQL statement.

77) **How do you record a data driven test?**

a) We can create a data-driven testing using data from a flat file, data table or a database.

   i. **Using Flat File**: we actually store the data to be used in a required format in the file. We access the file using the File manipulation commands, reads data from the file and assign the variables with data.

   ii. **Data Table:** It is an excel file. We can store test data in these files and manipulate them. We use the '**ddt_***' functions to manipulate data in the data table.

   iii. **Database:** we store test data in the database and access these data using '**db_***' functions.

78) **How do you convert a database file to a text file?**

a) You can use Data Junction to create a conversion file which converts a database to a target text file.

79) **How do you parameterize database check points?**

a) When you create a standard database checkpoint using ODBC (Microsoft Query), you can add parameters to an SQL statement to parameterize the checkpoint. This is useful if you want to create a database checkpoint with a query in which the SQL statement defining your query changes.

80) **How do you create parameterize SQL commands?**

a) A parameterized query is a query in which at least one of the fields of the WHERE clause is parameterized, i.e., the value of the field is specified by a question mark symbol (?). For example, the following SQL statement is based on a query on the database in the sample Flight Reservation application:

   i. *SELECT Flights.Departure, Flights.Flight_Number, Flights.Day_Of_Week FROM Flights Flights WHERE (Flights.Departure=?) AND (Flights.Day_Of_Week=?)*

SELECT defines the columns to include in the query.
FROM specifies the path of the database.
WHERE (optional) specifies the conditions, or filters to use in the query.
Departure is the parameter that represents the departure point of a flight.
Day_Of_Week is the parameter that represents the day of the week of a flight.

b) When creating a database checkpoint, you insert a db_check statement into your test script. When you parameterize the SQL statement in your checkpoint, the **db_check** function has a fourth, optional, argument: the **parameter_array** argument. A statement similar to the following is inserted into your test script:

db_check("list1.cdl", "dbvf1", NO_LIMIT, dbvf1_params);

The **parameter_array** argument will contain the values to substitute for the parameters in the parameterized checkpoint.

81) Explain the following commands:

a) db_connect
   i. to connect to a database
   db_connect(<session_name>, <connection_string>);

b) db_execute_query
   i. to execute a query
   db_execute_query (session_name, SQL, record_number);

   record_number is the out value.

c) db_get_field_value
   i. returns the value of a single field in the specified row_index and column in the session_name database session.

   db_get_field_value (session_name, row_index, column);

d) db_get_headers
   i. returns the number of column headers in a query and the content of the column headers, concatenated and delimited by tabs.

   db_get_headers (session_name, header_count, header_content);

e) db_get_row
   i. returns the content of the row, concatenated and delimited by tabs.

   db_get_row (session_name, row_index, row_content);

f) db_write_records
   i. writes the record set into a text file delimited by tabs.

db_write_records (session_name, output_file [, headers [, record_limit]]);

g) db_get_last_error
   i. returns the last error message of the last ODBC or Data Junction operation in the session_name database session.

db_get_last_error (session_name, error);

h) db_disconnect
   - **i.** disconnects from the database and ends the database session.

   db_disconnect (session_name);

i) db_dj_convert
   - **i.** runs the djs_file Data Junction export file. When you run this file, the Data Junction Engine converts data from one spoke (source) to another (target). The optional parameters enable you to override the settings in the Data Junction export file.

db_dj_convert (djs_file [, output_file [, headers [, record_limit]]]);

82) What check points you will use to read and check text on the GUI and explain its syntax?

a) You can use text checkpoints in your test scripts to read and check text in GUI objects and in areas of the screen. While creating a test you point to an object or a window containing text. WinRunner reads the text and writes a TSL statement to the test script. You may then add simple programming elements to your test scripts to verify the contents of the text.

b) You can use a text checkpoint to:
   - **i.** Read text from a GUI object or window in your application, using **obj_get_text** and **win_get_text**
   - **ii.** Search for text in an object or window, using **win_find_text** and **obj_find_text**
   - **iii.** Move the mouse pointer to text in an object or window, using **obj_move_locator_text** and **win_move_locator_text**
   - **iv.** Click on text in an object or window, using **obj_click_on_text** and **win_click_on_text**

83) Explain Get Text checkpoint from object/window with syntax?

a) We use *obj_get_text (<logical_name>, <out_text>)* function to get the text from an object

b) We use *win_get_text (window, out_text [, x1, y1, x2, y2])* function to get the text from a window.

84) Explain Get Text checkpoint from screen area with syntax?

a) We use *win_get_text (window, out_text [, x1, y1, x2, y2])* function to get the text from a window.

85) Explain Get Text checkpoint from selection (web only) with syntax?

a) Returns a text string from an object.

web_obj_get_text (object, table_row, table_column, out_text [, text_before, text_after, index]);

- **i.** *object* The logical name of the object.
- **ii.** *table_row* If the object is a table, it specifies the location of the row within a table. The string is preceded by the # character.
- **iii.** *table_column* If the object is a table, it specifies the location of the column within a table. The string is preceded by the # character.
- **iv.** *out_text* The output variable that stores the text string

v. *text_before* Defines the start of the search area for a particular text string.
vi. *text_after* Defines the end of the search area for a particular text string.
vii. *index* The occurrence number to locate. (The default parameter number is numbered 1).

86) Explain Get Text checkpoint web text checkpoint with syntax?

a) We use *web_obj_text_exists* function for web text checkpoints.

web_obj_text_exists (object, table_row, table_column, text_to_find [, text_before, text_after]);

a. *object* The logical name of the object to search.
b. *table_row* If the object is a table, it specifies the location of the row within a table. The string is preceded by the character #.
c. *table_column* If the object is a table, it specifies the location of the column within a table. The string is preceded by the character #.
d. *text_to_find* The string that is searched for.
e. *text_before* Defines the start of the search area for a particular text string.
f. *text_after* Defines the end of the search area for a particular text string.

87) Which TSL functions you will use for

a) Searching text on the window

i. *find_text (string, out_coord_array, search_area [, string_def]);*

string The string that is searched for. The string must be complete, contain no spaces, and it must be preceded and followed by a space outside the quotation marks. To specify a literal, case-sensitive string, enclose the string in quotation marks. Alternatively, you can specify the name of a string variable. In this case, the string variable can include a regular expression.

*out_coord_array*The name of the array that stores the screen coordinates of the text (see explanation below).

search_area The area to search, specified as coordinates x1,y1,x2,y2. These define any two diagonal corners of a rectangle. The interpreter searches for the text in the area defined by the rectangle.
string_def Defines the type of search to perform. If no value is specified, (0 or FALSE, the default), the search is for a single complete word only. When 1, or TRUE, is specified, the search is not restricted to a single, complete word.

b) getting the location of the text string

i. *win_find_text (window, string, result_array [, search_area [, string_def]]);*

window The logical name of the window to search.

string The text to locate. To specify a literal, case sensitive string, enclose the string in quotation marks. Alternatively, you can specify the name of a string variable. The value of the string variable can include a regular expression. The regular expression should not include an exclamation mark (!), however, which is treated as a literal character. For more information regarding Regular Expressions, refer to the "Using Regular Expressions" chapter in your User's Guide.

result_array The name of the output variable that stores the location of the string as a four-element array.

search_area The region of the object to search, relative to the window. This area is defined as a pair of coordinates, with x1,y1,x2,y2 specifying any two diagonally opposite corners of the rectangular search region. If this parameter is not defined, then the entire window is considered the search area.

string_def Defines how the text search is performed. If no string_def is specified, (0 or FALSE, the default parameter), the interpreter searches for a complete word only. If 1, or TRUE, is specified, the search is not restricted to a single, complete word.

c) Moving the pointer to that text string
   i. *win_move_locator_text (window, string [,search_area [,string_def]]);*

window The logical name of the window.

string The text to locate. To specify a literal, case sensitive string, enclose the string in quotation marks. Alternatively, you can specify the name of a string variable. The value of the string variable can include a regular expression (the regular expression need not begin with an exclamation mark).

search_area The region of the object to search, relative to the window. This area is defined as a pair of coordinates, with x1, y1, x2, y2 specifying any two diagonally opposite corners of the rectangular search region. If this parameter is not defined, then the entire window specified is considered the search area.

string_def Defines how the text search is performed. If no string_def is specified, (0 or FALSE, the default parameter), the interpreter searches for a complete word only. If 1, or TRUE, is specified, the search is not restricted to a single, complete word.

d) Comparing the text
   i. *compare_text (str1, str2 [, chars1, chars2]);*

str1, str2 The two strings to be compared.

chars1 One or more characters in the first string.

chars2 One or more characters in the second string. These characters are substituted for those in chars1.

88) What are the steps of creating a data driven test?

a) The steps involved in data driven testing are:
   i. Creating a test
   ii. Converting to a data-driven test and preparing a database
   iii. Running the test
   iv. Analyzing the test results.

89) Record a data driven test script using data driver wizard?

a) You can use the DataDriver Wizard to convert your entire script or a part of your script into a data-driven test. For example, your test script may include recorded operations, checkpoints, and other statements that do not need to be repeated for multiple sets of data. You need to parameterize only the portion of your test script that you want to run in a loop with multiple sets of data.

To create a data-driven test:

i. If you want to turn only part of your test script into a data-driven test, first select those lines in the test script.
ii. Choose Tools > DataDriver Wizard.
iii. If you want to turn only part of the test into a data-driven test, click Cancel. Select those lines in the test script and reopen the DataDriver Wizard. If you want to turn the entire test into a data-driven test, click Next.
iv. The **Use a new or existing Excel table** box displays the name of the Excel file that WinRunner creates, which stores the data for the data-driven test. Accept the default data table for this test, enter a different name for the data table, or use
v. The browse button to locate the path of an existing data table. By default, the data table is stored in the test folder.
vi. In the Assign a name to the variable box, enter a variable name with which to refer to the data table, or accept the default name, “table.”
vii. At the beginning of a data-driven test, the Excel data table you selected is assigned as the value of the table variable. Throughout the script, only the table variable name is used. This makes it easy for you to assign a different data table
viii. To the script at a later time without making changes throughout the script.
ix. Choose from among the following options:
    1. Add statements to create a data-driven test: Automatically adds statements to run your test in a loop: sets a variable name by which to refer to the data table; adds braces ({and}), a for statement, and a **ddt_get_row_count** statement to your test script selection to run it in a loop while it reads from the data table; adds **ddt_open** and **ddt_close** statements
    2. To your test script to open and close the data table, which are necessary in order to iterate rows in the table. Note that you can also add these statements to your test script manually.
    3. If you do not choose this option, you will receive a warning that your data-driven test must contain a loop and statements to open and close your datatable.
    4. Import data from a database: Imports data from a database. This option adds **ddt_update_from_db**, and **ddt_save** statements to your test script after the ddt_open statement.

5. Note that in order to import data from a database, either Microsoft Query or Data Junction must be installed on your machine. You can install Microsoft Query from the custom installation of Microsoft Office. Note that Data Junction is not automatically included in your WinRunner package. To purchase Data Junction, contact your Mercury Interactive representative. For detailed information on working with Data Junction, refer to the documentation in the Data Junction package.
6. **Parameterize the test**: Replaces fixed values in selected checkpoints and in recorded statements with parameters, using the **ddt_val** function, and in the data table, adds columns with variable values for the parameters. Line by line: Opens a wizard screen for each line of the selected test script, which enables you to decide whether to parameterize a particular line, and if so, whether to add a new column to the data table or use an existing column when parameterizing data.
7. **Automatically**: Replaces all data with ddt_val statements and adds new columns to the data table. The first argument of the function is the name of the column in the data table. The replaced data is inserted into the table.

x. The Test script line to parameterize box displays the line of the test script to parameterize. The highlighted value can be replaced by a parameter. The Argument to be replaced box displays the argument (value) that you can replace with a parameter. You can use the arrows to select a different argument to replace.

Choose whether and how to replace the selected data:

1. Do not replace this data: Does not parameterize this data.
2. An existing column: If parameters already exist in the data table for this test, select an existing parameter from the list.
3. A new column: Creates a new column for this parameter in the data table for this test. Adds the selected data to this column of the data table. The default name for the new parameter is the logical name of the object in the selected. TSL statement above. Accept this name or assign a new name.

xi. The final screen of the wizard opens.

1. If you want the data table to open after you close the wizard, select Show data table now.
2. To perform the tasks specified in previous screens and close the wizard, click Finish.
3. To close the wizard without making any changes to the test script, click Cancel.

90) What are the three modes of running the scripts?

a) WinRunner provides three modes in which to run tests—Verify, Debug, and Update. You use each mode during a different phase of the testing process.

   i. **Verify**

      1. Use the Verify mode to check your application.

   ii. **Debug**

      1. Use the Debug mode to help you identify bugs in a test script.

   iii. **Update**

      1. Use the Update mode to update the expected results of a test or to create a new expected results folder.

91) Explain the following TSL functions:

a) Ddt_open

   i. Creates or opens a datatable file so that WinRunner can access it.

Syntax: *ddt_open (data_table_name, mode);*

data_table_name The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table. The first row in the file contains the names of the parameters. This row is labeled row 0.

mode The mode for opening the data table: DDT_MODE_READ (read-only) or DDT_MODE_READWRITE (read or write).

b) Ddt_save

   i. Saves the information into a data file.

Syntax: *dt_save (data_table_name);*

data_table_name The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table.

c) Ddt_close

   i. Closes a data table file

Syntax: *ddt_close (data_table_name);*

data_table_name The name of the data table. The data table is a Microsoft Excel file or a tabbed text file. The first row in the file contains the names of the parameters.

d) Ddt_export

   i. Exports the information of one data table file into a different data table file.

Syntax: *ddt_export (data_table_namename1, data_table_namename2);*

data_table_namename1	The source data table filename.
data_table_namename2	The destination data table filename.

e) Ddt_show

   i. Shows or hides the table editor of a specified data table.

Syntax: *ddt_show (data_table_name [, show_flag]);*

data_table_name The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table.

show_flag The value indicating whether the editor should be shown (default=1) or hidden (0).

f) Ddt_get_row_count
   i. Retrieves the no. of rows in a data tables

Syntax: *ddt_get_row_count (data_table_name, out_rows_count);*

data_table_name The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table. The first row in the file contains the names of the parameters.

out_rows_count The output variable that stores the total number of rows in the data table.

g) ddt_next_row
   i. Changes the active row in a database to the next row

Syntax: *ddt_next_row (data_table_name);*

data_table_name The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table. The first row in the file contains the names of the parameters.

h) ddt_set_row
   i. Sets the active row in a data table.

Syntax: *ddt_set_row (data_table_name, row);*

data_table_name The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table. The first row in the file contains the names of the parameters. This row is labeled row 0.

row The new active row in the data table.

i) ddt_set_val
   i. Sets a value in the current row of the data table

Syntax: *ddt_set_val (data_table_name, parameter, value);*

data_table_name The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table. The first row in the file contains the names of the parameters. This row is labeled row 0.
parameter The name of the column into which the value will be inserted.
value The value to be written into the table.

j) ddt_set_val_by_row
   i. Sets a value in a specified row of the data table.

Syntax: *ddt_set_val_by_row (data_table_name, row, parameter, value);*

data_table_name The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table. The first row in the file contains the names of the parameters. This row is labeled row 0.

row The row number in the table. It can be any existing row or the current row number plus 1, which will add a new row to the data table.

parameter The name of the column into which the value will be inserted.

value The value to be written into the table.

k) ddt_get_current_row
   i. Retrieves the active row of a data table.

Syntax: *ddt_get_current_row (data_table_name, out_row);*

data_table_name The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table. The first row in the file contains the names of the parameters. This row is labeled row 0.

out_row The output variable that stores the active row in the data table.

l) ddt_is_parameter
   i. Returns whether a parameter in a datatable is valid

Syntax: *ddt_is_parameter (data_table_name, parameter);*

data_table_name The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table. The first row in the file contains the names of the parameters.

parameter The parameter name to check in the data table.

m) ddt_get_parameters
   i. Returns a list of all parameters in a data table.

Syntax: *ddt_get_parameters (table, params_list, params_num);*

table The pathname of the data table.
params_list This out parameter returns the list of all parameters in the data table, separated by tabs.
params_num This out parameter returns the number of parameters in params_list.

n) ddt_val
   i. Returns the value of a parameter in the active roe in a data table.

Syntax: *ddt_val (data_table_name, parameter);*

data_table_name The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table. The first row in the file contains the names of the parameters.

parameter The name of the parameter in the data table.

o) ddt_val_by_row
   i. Returns the value of a parameter in the specified row in a data table.

Syntax: *ddt_val_by_row (data_table_name, row_number, parameter);*

data_table_name The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table. The first row in the file contains the names of the parameters. This row is labeled row 0.

row_number The number of the row in the data table.

parameter The name of the parameter in the data table.

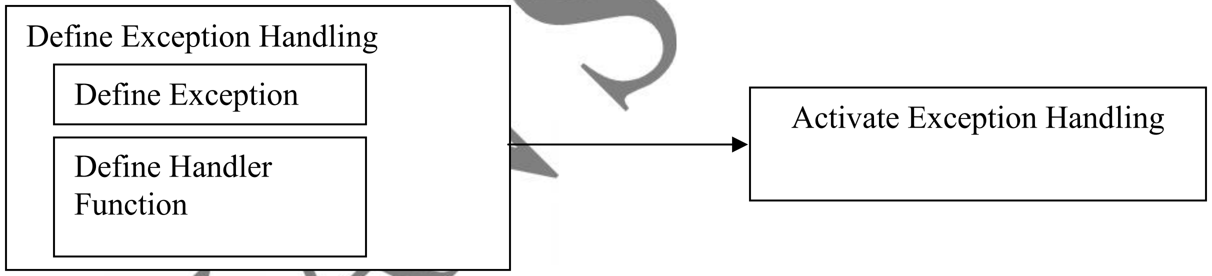
p) ddt_report_row
   i. Reports the active row in a data table to the test results
   Syntax: *ddt_report_row (data_table_name);*

data_table_name The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table. The first row in the file contains the names of the parameters. This row is labeled row 0.

q) ddt_update_from_db
   i. imports data from a database into a data table. It is inserted into your test script when you select the Import data from a database option in the DataDriver Wizard. When you run your test, this function updates the data table with data from the database.

92) How do you handle unexpected events and errors?

a) WinRunner uses exception handling to detect an unexpected event when it occurs and act to recover the test run.



WinRunner enables you to handle the following types of exceptions:

Pop-up exceptions: Instruct WinRunner to detect and handle the appearance of a specific window.

TSL exceptions: Instruct WinRunner to detect and handle TSL functions that return a specific error code.

Object exceptions: Instruct WinRunner to detect and handle a change in a property for a specific GUI object.

Web exceptions: When the WebTest add-in is loaded, you can instruct WinRunner to handle unexpected events and errors that occur in your Web site during a test run.

93) How do you handle pop-up exceptions?

a) A pop-up exception Handler handles the pop-up messages that come up during the execution of the script in the AUT. TO handle this type of exception we make WinRunner learn the window and also specify a handler to the exception. It could be

i. **Default actions**: WinRunner clicks the OK or Cancel button in the pop-up window, or presses Enter on the keyboard. To select a default handler, click the appropriate button in the dialog box.
ii. **User-defined handler**: If you prefer, specify the name of your own handler. Click User Defined Function Name and type in a name in the User Defined Function Name box.

94) How do you handle TSL exceptions?

a) A TSL exception enables you to detect and respond to a specific error code returned during test execution.
b) Suppose you are running a batch test on an unstable version of your application. If your application crashes, you want WinRunner to recover test execution. A TSL exception can instruct WinRunner to recover test execution by exiting the current test, restarting the application, and continuing with the next test in the batch.
c) The handler function is responsible for recovering test execution. When WinRunner detects a specific error code, it calls the handler function. You implement this function to respond to the unexpected error in the way that meets your specific testing needs.
d) Once you have defined the exception, WinRunner activates handling and adds the exception to the list of default TSL exceptions in the Exceptions dialog box. Default TSL exceptions are defined by the XR_EXCP_TSL configuration parameter in the wrun.ini configuration file.

95) How do you handle object exceptions?

a) During testing, unexpected changes can occur to GUI objects in the application you are testing. These changes are often subtle but they can disrupt the test run and distort results.
b) You could use exception handling to detect a change in property of the GUI object during the test run, and to recover test execution by calling a handler function and continue with the test execution

96) How do you comment your script?

a) We comment a script or line of script by inserting a '#' at the beginning of the line.

97) What is a compile module?

a) A compiled module is a script containing a library of user-defined functions that you want to call frequently from other tests. When you load a compiled module, its functions are automatically compiled and remain in memory. You can call them directly from within any test.
b) Compiled modules can improve the organization and performance of your tests. Since you debug compiled modules before using them, your tests will require less error-checking. In addition, calling a function that is already compiled is significantly faster than interpreting a function in a test script.

98) What is the difference between script and compile module?

a) Test script contains the executable file in WinRunner while Compiled Module is used to store reusable functions. Complied modules are not executable.
b) WinRunner performs a pre-compilation automatically when it saves a module assigned a property value of "Compiled Module".

c) By default, modules containing TSL code have a property value of "main". Main modules are called for execution from within other modules. Main modules are dynamically compiled into machine code only when WinRunner recognizes a "call" statement. Example of a call for the "app_init" script:

```
call cso_init();
call( "C:\\MyAppFolder\\" & "app_init" );
```

d) Compiled modules are loaded into memory to be referenced from TSL code in any module. Example of a load statement:

```
reload (“C:\\MyAppFolder\\" & "flt_lib");
or
load ("C:\\MyAppFolder\\" & "flt_lib");
```

99) Write and explain various loop command?

a) A for loop instructs WinRunner to execute one or more statements a specified number of times.

It has the following **syntax:**

for ([expression1]; [expression2]; [expression3])
statement

i. First, expression1 is executed. Next, expression2 is evaluated. If expression2 is true, statement is executed and expression3 is executed. The cycle is repeated as long as expression2 remains true. If expression2 is false, the for statement terminates and execution passes to the first statement immediately following.

ii. For example, the for loop below selects the file UI_TEST from the File Name list

iii. in the Open window. It selects this file five times and then stops.

```
set_window ("Open")
for (i=0; i<5; i++)
  list_select_item("File_Name:_1","UI_TEST"); #Item Number2
```

b) A while loop executes a block of statements for as long as a specified condition is true.

It has the following syntax:

while (expression)
statement ;

i. While expression is true, the statement is executed. The loop ends when the expression is false. For example, the while statement below performs the same function as the for loop above.

```
set_window ("Open");
i=0;
while (i<5){
        i++;
        list_select_item ("File Name:_1", "UI_TEST"); # Item
        Number 2
}
```

c) A do/while loop executes a block of statements for as long as a specified condition is true. Unlike the for loop and while loop, a do/while loop tests the conditions at the end of the loop, not at the beginning.

A do/while loop has the following syntax:

```
do
        statement
while (expression);
```

i. The statement is executed and then the expression is evaluated. If the expression is true, then the cycle is repeated. If the expression is false, the cycle is not repeated.

ii. For example, the do/while statement below opens and closes the Order dialog box of Flight Reservation five times.

```
set_window ("Flight Reservation");
i=0;
do
{
        menu_select_item ("File;Open Order...");
        set_window ("Open Order");
        button_press ("Cancel");
        i++;
}
while (i<5);
```

100) Write and explain decision making command?

a) You can incorporate decision-making into your test scripts using if/else or switch statements.

i. An if/else statement executes a statement if a condition is true; otherwise, it executes another statement.
It has the following syntax:

```
if ( expression )
        statement1;
[ else
statement2; ]
```

expression is evaluated. If expression is true, statement1 is executed. If expression1 is false, statement2 is executed.

b) A switch statement enables WinRunner to make a decision based on an expression that can have more than two values.

It has the following syntax:

```
switch (expression )
{
        case case_1: statements
        case case_2: statements
        case case_n: statements
        default: statement(s)
}
```

The switch statement consecutively evaluates each case expression until one is found that equals the initial expression. If no case is equal to the expression, then the default statements are executed. The default statements are optional.

101) Write and explain switch command?

a) A switch statement enables WinRunner to make a decision based on an expression that can have more than two values.

It has the following syntax:

```
switch (expression )
{
    case case_1:  statements
    case case_2: statements
    case case_n: statements
    default: statement(s)
}
```

b) The switch statement consecutively evaluates each case expression until one is found that equals the initial expression. If no case is equal to the expression, then the default statements are executed. The default statements are optional.

102) How do you write messages to the report?

a) To write message to a report we use the report_msg statement
Syntax: *report_msg (message);*

103) What is a command to invoke application?

a) Invoke_application is the function used to invoke an application.
Syntax: *invoke_application(file, command_option, working_dir, SHOW);*

104) What is the purpose of tl_step command?

a) Used to determine whether sections of a test pass or fail.
Syntax: *tl_step(step_name, status, description);*

105) Which TSL function you will use to compare two files?

a) We can compare 2 files in WinRunner using the file_compare function.
Syntax: *file_compare (file1, file2 [, save file]);*

106) What is the use of function generator?

a) The Function Generator provides a quick, error-free way to program scripts. You can:
   i. Add Context Sensitive functions that perform operations on a GUI object or get information from the application being tested.
   ii. Add Standard and Analog functions that perform non-Context Sensitive tasks such as synchronizing test execution or sending user-defined messages to a report.
   iii. Add Customization functions that enable you to modify WinRunner to suit your testing environment.

107) What is the use of putting call and call_close statements in the test script?

a) You can use two types of call statements to invoke one test from another:
   i. A call statement invokes a test from within another test.
   ii. A call_close statement invokes a test from within a script and closes the test when the test is completed.

iii. The call statement has the following syntax:
    1. call test_name ([parameter1, parameter2, ...parametern]);

iv. The call_close statement has the following syntax:
    1. call_close test_name ([parameter1, parameter2, ... parametern]);

v. The test_name is the name of the test to invoke. The parameters are the parameters defined for the called test.

vi. The parameters are optional. However, when one test calls another, the call statement should designate a value for each parameter defined for the called test. If no parameters are defined for the called test, the call statement must contain an empty set of parentheses.

108) What is the use of treturn and texit statements in the test script?

a) The treturn and texit statements are used to stop execution of called tests.
   i. The treturn statement stops the current test and returns control to the calling test.
   ii. The texit statement stops test execution entirely, unless tests are being called from a batch test. In this case, control is returned to the main batch test.

b) Both functions provide a return value for the called test. If treturn or texit is not used, or if no value is specified, then the return value of the call statement is 0.

treturn

c) The treturn statement terminates execution of the called test and returns control to the calling test.
The syntax is:
    treturn [(expression)];

d) The optional expression is the value returned to the call statement used to invoke the test.

texit

e) When tests are run interactively, the texit statement discontinues test execution. However, when tests are called from a batch test, texit ends execution of the current test only; control is then returned to the calling batch test.
The syntax is:
    texit [(expression)];

109) Where do you set up the search path for a called test.

a) The search path determines the directories that WinRunner will search for a called test.

b) To set the search path, choose Settings > General Options. The General Options dialog box opens. Click the Folders tab and choose a search path in the Search Path for Called Tests box. WinRunner searches the directories in the order in which they are listed in the box. Note that the search paths you define remain active in future testing sessions.

110) How you create user-defined functions and explain the syntax?

a) A user-defined function has the following structure:

```
[class] function name ([mode] parameter...)
{
        declarations;
        statements;
}
```

b) The class of a function can be either static or public. A static function is available only to the test or module within which the function was defined.

c) Parameters need not be explicitly declared. They can be of mode in, out, or inout. For all non-array parameters, the default mode is in. For array parameters, the default is inout. The significance of each of these parameter types is as follows:

   in: A parameter that is assigned a value from outside the function.
   out: A parameter that is assigned a value from inside the function.
   inout: A parameter that can be assigned a value from outside or inside the function.

111) What does static and public class of a function means?

a) The class of a function can be either static or public.
b) A static function is available only to the test or module within which the function was defined.
c) Once you execute a public function, it is available to all tests, for as long as the test containing the function remains open. This is convenient when you want the function to be accessible from called tests. However, if you want to create a function that will be available to many tests, you should place it in a compiled module. The functions in a compiled module are available for the duration of the testing session.
d) If no class is explicitly declared, the function is assigned the default class, public.

112) What does in, out and input parameters means?

a) in: A parameter that is assigned a value from outside the function.
b) out: A parameter that is assigned a value from inside the function.
c) inout: A parameter that can be assigned a value from outside or inside the function.

113) What is the purpose of return statement?

a) This statement passes control back to the calling function or test. It also returns the value of the evaluated expression to the calling function or test. If no expression is assigned to the return statement, an empty string is returned.
   Syntax: *return [(expression)];*

114) What does auto, static, public and extern variables means?

a) **auto**: An auto variable can be declared only within a function and is local to that function. It exists only for as long as the function is running. A new copy of the variable is created each time the function is called.
b) **static**: A static variable is local to the function, test, or compiled module in which it is declared. The variable retains its value until the test is terminated by an Abort command. This variable is initialized each time the definition of the function is executed.

c) **public**: A public variable can be declared only within a test or module, and is available for all functions, tests, and compiled modules.
d) **extern**: An extern declaration indicates a reference to a public variable declared outside of the current test or module.

115) How do you declare constants?

a) The const specifier indicates that the declared value cannot be modified. The class of a constant may be either public or static. If no class is explicitly declared, the constant is assigned the default class public. Once a constant is defined, it remains in existence until you exit WinRunner.
b) The syntax of this declaration is:
[class] const name [= expression];

116) How do you declare arrays?

a) The following syntax is used to define the class and the initial expression of an array. Array size need not be defined in TSL.
b) class array_name [ ] [=init_expression]
c) The array class may be any of the classes used for variable declarations (auto, static, public, extern).

117) How do you load and unload a compile module?

a) In order to access the functions in a compiled module you need to load the module. You can load it from within any test script using the load command; all tests will then be able to access the function until you quit WinRunner or unload the compiled module.
b) You can load a module either as a system module or as a user module. A system module is generally a closed module that is "invisible" to the tester. It is not displayed when it is loaded, cannot be stepped into, and is not stopped by a pause command. A system module is not unloaded when you execute an unload statement with no parameters (global unload).
load (module_name [,1|0] [,1|0]);

The *module_name* is the name of an existing compiled module.

Two additional, optional parameters indicate the type of module. The first parameter indicates whether the function module is a system module or a user module: 1 indicates a system module; 0 indicates a user module.

(Default = 0)

The second optional parameter indicates whether a user module will remain open in the WinRunner window or will close automatically after it is loaded: 1 indicates that the module will close automatically; 0 indicates that the module will remain open.
(Default = 0)
c) The unload function removes a loaded module or selected functions from memory.
d) It has the following syntax:
unload ([module_name | test_name [, "function_name"]]);

118) Why you use reload function?

a) If you make changes in a module, you should reload it. The reload function removes a loaded module from memory and reloads it (combining the functions of unload and load).
The syntax of the reload function is:

reload (module_name [,1|0] [,1|0]);

The module_name is the name of an existing compiled module.

Two additional optional parameters indicate the type of module. The first parameter indicates whether the module is a system module or a user module: 1 indicates a system module; 0 indicates a user module.
(Default = 0)
The second optional parameter indicates whether a user module will remain open in the WinRunner window or will close automatically after it is loaded. 1 indicates that the module will close automatically. 0 indicates that the module will remain open.
(Default = 0)

119) Why does the minus sign not appear when using obj_type(), win_type(), type()?

If using any of the type() functions, minus signs actually means hold down the button for the previous character. The solution is to put a backslash character "\\" before the minus sign. This also applies to + < >.

120) Write and explain compile module?

121) How do you call a function from external libraries (dll).
122) What is the purpose of load_dll?
123) How do you load and unload external libraries?
124) How do you declare external functions in TSL?
125) How do you call windows APIs, explain with an example?
126) Write TSL functions for the following interactive modes:
   i. Creating a dialog box with any message you specify, and an edit field.
   ii. Create dialog box with list of items and message.
   iii. Create dialog box with edit field, check box, and execute button, and a cancel button.
   iv. Creating a browse dialog box from which user selects a file.
   v. Create a dialog box with two edit fields, one for login and another for password input.
127) What is the purpose of step, step into, step out, step to cursor commands for debugging your script?
128) How do you update your expected results?
129) How do you run your script with multiple sets of expected results?
130) How do you view and evaluate test results for various check points?
131) How do you view the results of file comparison?
132) What is the purpose of Wdiff utility?
133) What are batch tests and how do you create and run batch tests ?
134) How do you store and view batch test results?
135) How do you execute your tests from windows run command?
136) Explain different command line options?
137) What TSL function you will use to pause your script?
138) What is the purpose of setting a break point?
139) What is a watch list?
140) During debugging how do you monitor the value of the variables?