**UFT Assignment Questions**

### **1. Introduction to UFT:**

**Q1: What is UFT (Unified Functional Testing)? How is it different from other test automation tools like Selenium or QTP?**

Unified Functional Testing (UFT) is a comprehensive test automation tool developed by Micro Focus, primarily used for functional and regression testing of applications. It supports a wide range of technologies, including web, desktop, and mobile applications, and provides both record-and-playback capabilities and script-based testing with the help of VBScript. UFT integrates with other testing tools, enabling test management and reporting across different platforms.

What sets UFT apart from other test automation tools like Selenium or QTP (QuickTest Professional, the predecessor of UFT) is its deep integration with various enterprise-level tools and its built-in support for testing both web and non-web applications.

**Differences:**

1. Language and Scripting:

UFT uses VBScript for scripting, which is simple and easy to learn, especially for users familiar with VB-based languages.

Selenium, on the other hand, supports multiple programming languages like Java, Python, and C#, offering more flexibility in scripting.

1. Supported Technologies:

UFT provides extensive out-of-the-box support for desktop, web, and mobile applications, including legacy systems.

Selenium is mainly focused on web applications and relies heavily on third-party tools for broader test coverage.

1. Cost:

UFT is a commercial tool with a licensing fee, making it suitable for large enterprises with a dedicated testing budget.

Selenium is open-source and free, making it a preferred choice for smaller teams or projects with a limited budget.

1. Ease of Use:

UFT’s record-and-playback feature allows non-technical users to quickly create test scripts, which can be advantageous in rapid testing environments.

Selenium often requires more coding knowledge and setup, but it offers more control over the test scenarios.

1. Integration:

UFT integrates well with other Micro Focus products, including ALM (Application Lifecycle Management) for test management.

Selenium’s integrations are often with third-party CI/CD tools, offering flexibility but requiring additional configuration.

**Q2: List the key features of UFT. Explain how it supports functional, regression, and GUI testing.**

Unified Functional Testing (UFT), formerly known as QTP (QuickTest Professional), is a comprehensive test automation tool developed by Micro Focus. It supports a wide range of testing capabilities and is known for its versatility and integration features. Here are the key features of UFT:

1. Keyword-Driven and Scripted Testing: UFT allows both keyword-driven testing, where testers create tests using a table of keywords, and scripted testing, where detailed test scripts are written in VBScript.
2. Cross-Browser Support: UFT supports testing across different browsers, including Internet Explorer, Chrome, Firefox, and Edge, enabling seamless web application testing.
3. Integrated Test Management: UFT integrates with test management tools like ALM (Application Lifecycle Management), allowing for efficient test planning, execution, and reporting.
4. Support for Multiple Technologies: UFT supports testing for various applications, including web, desktop, mobile, and SAP applications. It provides deep integration with technologies such as .NET, Java, Oracle, and more.
5. Object Recognition and Identification: UFT uses robust object recognition techniques, including descriptive programming, to identify and interact with UI elements, ensuring stability even when the UI changes.
6. Data-Driven Testing: It supports data-driven testing by connecting test scripts to data sources (e.g., Excel, database) to execute tests with different data sets, improving test coverage.
7. Regression Testing: UFT's ability to reuse test scripts makes it an excellent tool for regression testing, ensuring that existing functionality works as expected after code changes.
8. Integration with CI/CD: UFT integrates with continuous integration tools such as Jenkins, allowing automated tests to run as part of the CI/CD pipeline.
9. Multi-platform Support: UFT supports testing on different operating systems, including Windows and macOS, for applications running on both platforms.

Support for Functional, Regression, and GUI Testing:

1. Functional Testing: UFT allows testers to validate the functionality of applications by automating user interactions. Its ability to interact with various UI elements (buttons, dropdowns, text fields) ensures that the application behaves as expected under different scenarios.
2. Regression Testing: UFT excels in regression testing by enabling the reuse of existing test scripts across different versions of the application. This minimizes the time and effort spent on creating new test cases while ensuring that new code changes do not introduce defects in previously working functionality.
3. GUI Testing: UFT automates GUI testing by simulating user actions on graphical user interfaces and validating whether the UI elements function correctly. It also offers detailed reporting on GUI behavior, ensuring consistency and visual correctness across platforms.

**Q3: What are the different types of objects that UFT can recognize? Give examples of each type.**

In UFT (Unified Functional Testing), objects are elements of the application under test that UFT can identify, interact with, and manipulate. These objects fall into different categories based on their type and behavior. Here are the key types of objects that UFT can recognize, along with examples:

1. Standard Windows Controls  
    These are common elements found in Windows-based applications.  
   Example :: Button, TextBox, ListBox, CheckBox.
2. Web Objects  
    These are elements in web-based applications, including HTML tags.  
   Example :Button (HTML), TextField (HTML), Link, Dropdown.
3. Virtual Objects  
    UFT can create virtual objects to recognize non-standard or custom objects that the default object recognition does not capture.  
   Example :Custom image buttons, unique graphics, or custom controls that don't have standard object recognition.
4. ActiveX Controls  
    These are elements that are part of the ActiveX framework, typically in desktop applications.  
   Example :ActiveX buttons, custom ActiveX controls used in legacy systems.
5. Mobile Objects  
    For mobile applications, UFT identifies objects in both native and hybrid apps.  
   Example :Button (iOS/Android), TextField (iOS/Android), ListItem.
6. SAP Objects  
    UFT has built-in support for SAP GUI applications, identifying objects specific to SAP systems.  
   Example :SAP Table, SAP Button, SAP TextField.
7. Java Objects  
    Java-based applications are supported by UFT, allowing it to interact with Java objects.  
   Example :JavaButton, JavaTextField, JavaListBox.
8. Flex Objects  
    UFT can recognize and interact with Flex-based objects used in Adobe Flash applications.

Example :FlexButton, FlexTextField, FlexComboBox.

1. WPF (Windows Presentation Foundation) Objects  
    For modern desktop applications built using WPF, UFT can identify specific controls.  
    Example : WPF Button, WPF TextBox, WPF ComboBox.

### **2. Creating and Running a Basic Test in UFT:**

**Q4: Create a simple test in UFT to open the Notepad application, type a text message, and save the file. Include the steps to record and run the test.**

**Steps to Record and Run the Test in UFT:**

1. Start UFT and Create a New Test:

* Open UFT (Unified Functional Testing).
* Click on File > New > Test.
* Choose GUI Test and give it a name (e.g., NotepadTest).
* Click OK.

2. Record the Test:

* Click the Record button in the toolbar or use the shortcut Ctrl+R.
* In the Record and Run Settings, choose Desktop as the environment.
* Click OK to start recording.

3. Record Steps:

* Open Notepad:

Go to Start Menu > All Programs > Accessories > Notepad to open it.

UFT will automatically record this step.

* Type Text in Notepad:

In Notepad, type a text message (e.g., "Hello, this is a test").

UFT will record the actions of typing the message.

* Save the File:

Click on File > Save As.

Enter a file name (e.g., TestFile.txt), select a location, and click Save.

UFT will record this as well.

4. Stop the Recording:

* Click the Stop button on the UFT toolbar or press Ctrl+Shift+R to stop recording.
* UFT will generate the test script based on the actions you performed.

Test Script Example:

Here’s the script that would be generated after recording the steps:

' Open Notepad

SystemUtil.Run "notepad.exe"

' Wait for Notepad to open and become active

Window("Notepad").WinButton("OK").WaitProperty "enabled", True, 10

' Type a message in Notepad

Window("Notepad").Type "Hello, this is a test"

' Save the file

Window("Notepad").MenuSelect "File->SaveAs"

Window("Save As").WinEdit("Edit").Set "C:\TestFile.txt"

Window("Save As").WinButton("Save").Click

' Close Notepad

Window("Notepad").Close

**Explanation of the Script:**

1. **SystemUtil.Run "notepad.exe"**: Opens the Notepad application.
2. **Window("Notepad").WinButton("OK").WaitProperty "enabled", True, 10**: Waits for the Notepad window to be ready.
3. **Window("Notepad").Type "Hello, this is a test"**: Types the text message in Notepad.
4. **Window("Notepad").MenuSelect "File->SaveAs"**: Opens the "Save As" dialog.
5. **Window("Save As").WinEdit("Edit").Set "C:\TestFile.txt"**: Sets the file name in the "Save As" dialog.
6. **Window("Save As").WinButton("Save").Click**: Clicks the "Save" button to save the file.
7. **Window("Notepad").Close**: Closes the Notepad application.

**5. Run the Test:**

* After recording the test, you can **run** it by clicking the **Run** button in the toolbar or pressing F5.
* UFT will execute the steps in the script and perform the actions on your machine, opening Notepad, typing the message, saving the file, and closing the application.

**6. Verify the Output:**

* After the test has run, check the location (C:\TestFile.txt) to verify the file was created and contains the correct message.
* You can also review the **Test Results** in UFT to ensure that all actions were performed successfully.

**Q5: Write a simple UFT script to open a web browser, navigate to a website (e.g., www.google.com), and perform a Google search.**

' Open a web browser and navigate to www.google.com

SystemUtil.Run "iexplore.exe", "http://www.google.com"

' Wait for the Google page to load

Browser("Google").Page("Google").Sync

' Find the search box and type a query (e.g., "UFT automation")

Browser("Google").Page("Google").WebEdit("q").Set "UFT automation"

' Click the Google Search button

Browser("Google").Page("Google").WebButton("Google Search").Click

' Wait for the results page to load

Browser("Google").Page("Google").Sync

' Close the browser

Browser("Google").Close

1. **Browser("Google").Page("Google").Sync**: Waits for the search results page to load completely.
2. **Browser("Google").Close**: This closes the browser after the search.

**Running the Script:**

* After writing the script in UFT, click the **Run** button or press F5 to execute it.
* UFT will open the browser, navigate to **Google**, search for "UFT automation," and close the browser after the results are displayed.

### **3. Object Repository and Object Identification:**

**Q6: What is an object repository in UFT? Explain the difference between "Local Object Repository" and "Shared Object Repository."**

In UFT (Unified Functional Testing), an Object Repository is a centralized storage location that contains objects and their properties used in test scripts. These objects represent the elements in the application under test (AUT), such as buttons, text boxes, and links. The Object Repository allows for easy identification and interaction with these objects during test execution.

Difference

1. **Local Object Repository:**

Scope: It is specific to a single test script.

Usage: The objects are stored and used only within that particular test, making it more suitable for individual or isolated tests.

Flexibility: Any modifications made to the objects affect only the test in which they reside.

1. **Shared Object Repository:**

Scope: It is accessible across multiple test scripts or actions within a test.

Usage: This is ideal when multiple tests need to interact with the same objects, ensuring consistency and reducing redundancy in object management.

Collaboration: Changes made to a Shared Object Repository are reflected across all tests using that repository, promoting better maintainability.

**Q7: Explain the concept of "Object Identification" in UFT. How does UFT recognize objects on the application being tested?**

In UFT (Unified Functional Testing), Object Identification refers to the process by which UFT recognizes and interacts with various elements of the application being tested, such as buttons, text fields, and links. UFT identifies these objects using a combination of properties and methods to uniquely distinguish them from other elements within the application.

In UFT (Unified Functional Testing), Object Identification refers to the process by which UFT recognizes and interacts with various elements of the application being tested, such as buttons, text fields, and links. UFT identifies these objects using a combination of properties and methods to uniquely distinguish them from other elements within the application.

UFT uses two primary approaches for object identification:

1. Static Object Identification: UFT identifies objects based on a fixed set of properties, such as the object’s name, class, or ID. These properties do not change between tests, making this method reliable for stable applications.
2. Dynamic Object Identification: UFT can also adapt to objects whose properties might change during runtime (e.g., dynamic IDs). It uses a dynamic set of properties, along with regular expressions or wildcard characters, to handle variations in object identification.

UFT maintains a Object Repository, where it stores objects and their corresponding properties. When a test is run, UFT compares the properties in the Object Repository with the objects present in the application to identify and interact with them. This process ensures that the script can execute correctly, even as the application evolves or objects change dynamically.

This method of object recognition allows UFT to automate tests effectively, ensuring consistent and accurate results.

ocess ensures that the script can execute correctly, even as the application evolves or objects change dynamically.

This method of object recognition allows UFT to automate tests effectively, ensuring consistent and accurate results.

### **4. Checkpoints and Verification:**

**Q10: What are checkpoints in UFT? Write a script to add a "Text Checkpoint" to verify that a specific text appears on a web page.**

**Checkpoints in UFT (Unified Functional Testing):**

Checkpoints in UFT are used to verify whether the application under test behaves as expected. They are inserted into the test scripts to validate specific attributes, content, or conditions in the application. Checkpoints allow UFT to compare the actual results with the expected results.

The types of checkpoints in UFT include:

1. **Text Checkpoint**: Verifies whether a specific text is present in a web page or object.
2. **Image Checkpoint**: Compares an image in the application with a stored reference image.
3. **Standard Checkpoint**: Verifies the properties of an object in the application.
4. **Database Checkpoint**: Verifies data in a database.
5. **Page Checkpoint**: Verifies the properties and content of a webpage.
6. **Table Checkpoint**: Verifies data in a table on a web page or application.

**Script to Add a "Text Checkpoint" in UFT:**

To add a "Text Checkpoint" to verify that a specific text appears on a web page, here is a simple script:

' Create a new instance of the browser

Dim Browser

Set Browser = Browser("micClass:=Browser")

' Navigate to a webpage

Browser.Navigate "http://example.com" ' Replace with the actual URL

' Wait for the page to load

Browser.Wait

' Add the Text Checkpoint

' Verifying that the text "Welcome to Example" is present on the page

Browser.Page("micClass:=Page").TextCheckPoint "Welcome to Example"

' Check the result of the checkpoint

If Browser.Page("micClass:=Page").Check(CheckPoint("Text Checkpoint")) Then

MsgBox "Checkpoint Passed: Text 'Welcome to Example' is found on the page."

Else

MsgBox "Checkpoint Failed: Text 'Welcome to Example' is not found on the page."

End If

' Close the browser

Browser.Close

**Q11: Explain the difference between "Standard Checkpoints" and "Database Checkpoints" in UFT. Give an example of when you would use each.**

In UFT (Unified Functional Testing), "Standard Checkpoints" and "Database Checkpoints" are both used to verify the correctness of an application during automated testing, but they serve different purposes.

1. Standard Checkpoints:

A Standard Checkpoint is used to verify the properties of an object or a screen in the application under test. It checks if the actual value of an object matches the expected value during a test run.

Example: If you want to verify that the "Submit" button on a web page is enabled, you can use a Standard Checkpoint to check the button's properties like its enabled state or label text.

1. Database Checkpoints:

A Database Checkpoint is used to verify the contents of a database query result, ensuring that the data returned from a database matches the expected values. It allows for validation of data in the back-end system, directly comparing the query result with expected values.

Example: If you are testing an online shopping system and want to ensure that after a purchase, the inventory in the database is updated correctly, you can use a Database Checkpoint to validate the quantity of a product in the database.

**Q12: How can you handle dynamic objects using UFT? Explain with an example of handling dynamic buttons that change text based on user interactions.**

In UFT (Unified Functional Testing), dynamic objects are those that change attributes (like text, name, ID, etc.) during runtime. Handling dynamic objects can be challenging because their properties are not fixed and might change frequently. To handle dynamic objects effectively in UFT, we can use **dynamic object identification** techniques such as **descriptive programming**, **regular expressions**, or **properties that remain constant** even if other attributes change.

**Handling Dynamic Buttons in UFT**

To handle dynamic buttons (e.g., buttons that change text based on user interactions), we can use **descriptive programming** with **regular expressions** or **properties that remain constant** (like name, id, class, etc.) to locate the buttons.

**Example: Handling Dynamic Buttons**

Consider a scenario where a button's text changes after the user clicks it, such as a "Start" button changing to "Stop" after the first click. Here’s how you can handle such a dynamic button using UFT:

**Descriptive Programming with Dynamic Properties**

In UFT, instead of relying on static object repositories, you can use descriptive programming. This allows you to define the properties of the object dynamically at runtime.

' Create a new instance of the browser and navigate to the webpage

Dim Browser

Set Browser = Browser("micClass:=Browser")

Browser.Navigate "http://example.com" ' Replace with your actual URL

' Wait for the page to load

Browser.Wait

' Define a variable for the button

Dim DynamicButton

' Handle dynamic button: Use a regular expression for the button text

Set DynamicButton = Browser.Page("micClass:=Page").WebButton("text:=Start.\*")

' Click the button if it exists

If DynamicButton.Exists(5) Then

DynamicButton.Click

MsgBox "Button clicked successfully!"

Else

MsgBox "Dynamic button not found!"

End If

' Handle the dynamic change (after clicking, the button text changes)

Set DynamicButton = Browser.Page("micClass:=Page").WebButton("text:=Stop.\*")

' Wait for the button text to change (optional, depending on the application)

Browser.Wait

' Verify if the button text has changed and click it again

If DynamicButton.Exists(5) Then

DynamicButton.Click

MsgBox "Button clicked again successfully!"

Else

MsgBox "Button text did not change."

End If

' Close the browser

Browser.Close

**Explanation:**

1. **Descriptive Programming**: We use descriptive programming (WebButton("text:=Start.\*")) to find the button based on its **text** property, using a regular expression (Start.\*). This regular expression matches any text that starts with "Start" but can end with any other characters (e.g., "Start", "Start123", etc.).
2. **Click the Button**: The script checks whether the button exists (DynamicButton.Exists(5)) and clicks it if found.
3. **Dynamic Text Change**: After clicking the button, we assume the text will change to "Stop". We then update the descriptive programming to match the new text (WebButton("text:=Stop.\*")).
4. **Verification**: The script then verifies if the button with the new text "Stop" exists and clicks it again.
5. **Wait for Dynamic Changes**: The Browser.Wait method ensures the script waits for the button’s text to change before proceeding.

### **5. Parameterization:**

**Q13: What is parameterization in UFT? Why is it important for automating tests? Demonstrate how to parameterize a test using input data (e.g., user credentials for a login page).**

**What is Parameterization in UFT?**

Parameterization in UFT (Unified Functional Testing) is the process of using variable values in test scripts instead of hardcoding them. This allows the test to run multiple times with different inputs, making the test more flexible and reusable. By parameterizing tests, you can test the same functionality with a variety of inputs, such as user credentials, product names, or search terms, without rewriting the entire test script.

**Why is Parameterization Important?**

* **Reusability**: The same test can be run with different sets of data without modifying the script each time.
* **Data-Driven Testing**: It allows testers to validate the application with different input values, ensuring robustness and accuracy.
* **Efficiency**: Reduces the need for manual test creation for different input data, speeding up the testing process.
* **Better Test Coverage**: Ensures the application behaves as expected under different conditions or inputs.

**How to Parameterize a Test in UFT (Example: Login Page)**

Let’s consider a simple example where we need to test a login page with different user credentials (username and password). Instead of hardcoding values, we will use a data table to input different sets of values for the login process.

**Steps to Parameterize a Test Using Input Data:**

1. **Create a Data Table**: You can use the UFT Data Table feature, which is similar to an Excel sheet, where you can store the test data (user credentials).
2. **Referencing the Data Table in the Script**: Use the DataTable functions in UFT to access the test data from the Data Table and pass it into the test.

**Step-by-Step Example:**

1. **Create a Data Table (Example: UserCredentials)**:

In UFT, you can create a Data Table by going to the "Data" tab and adding a new test data sheet (e.g., "UserCredentials").

The table should have columns for the username and password, like this:

| **Username** | **Password** |
| --- | --- |
| user1 | pass1 |
| user2 | pass2 |
| user3 | pass3 |

1. **Script to Parameterize the Login Test**:

' Set up the browser and navigate to the login page

Dim Browser

Set Browser = Browser("micClass:=Browser")

Browser.Navigate "http://example.com/login" ' Replace with actual URL

' Wait for the page to load

Browser.Wait

' Loop through the data table to get different user credentials

For i = 1 To DataTable.GetRowCount

' Retrieve username and password from the Data Table

Dim Username, Password

Username = DataTable.Value("Username", i)

Password = DataTable.Value("Password", i)

' Set the values in the login fields and submit the form

Browser.Page("micClass:=Page").WebEdit("name:=username").Set Username

Browser.Page("micClass:=Page").WebEdit("name:=password").Set Password

Browser.Page("micClass:=Page").WebButton("name:=login").Click

' Verify if the login is successful (you can add a checkpoint here to check success)

If Browser.Page("micClass:=Page").WebElement("text:=Welcome").Exists Then

MsgBox "Login successful for " & Username

Else

MsgBox "Login failed for " & Username

End If

' Log out if required (for this example, assuming a logout process)

' Browser.Page("micClass:=Page").WebButton("name:=logout").Click

Next

' Close the browser

Browser.Close

**Explanation of the Script:**

1. **Data Table Access**: The DataTable.Value("Username", i) retrieves the value of the username from the data table for the ith row, and similarly for the password.
2. **Looping Through the Data Table**: The For loop iterates over each row in the data table, allowing the test to run for different combinations of usernames and passwords.
3. **Setting Input Values**: The WebEdit("name:=username").Set Username and WebEdit("name:=password").Set Password commands set the values of the username and password input fields on the web page using the values retrieved from the data table.
4. **Login Verification**: The script checks if the login was successful by verifying the presence of a "Welcome" text on the page. You can replace this with any element that indicates successful login.
5. **Running the Test**: The test will run once for each row of data in the data table, testing different user credentials for login.

**Q14: Create a test that accepts input parameters (e.g., username and password) from an Excel file and performs a login using that data.**

**Steps to Create the Test:**

**1. Prepare the Excel File:**

* Create an Excel file (e.g., LoginData.xlsx) with columns for Username and Password.
* Example of data in LoginData.xlsx:

| **Username** | **Password** |
| --- | --- |
| user1 | pass123 |
| user2 | pass456 |
| user3 | pass789 |

**2. UFT Script to Read Data from Excel and Perform Login:**

Here’s a sample UFT script that reads data from the Excel file and performs a login on a web application:

' Declare variables

Dim xlApp, xlBook, xlSheet

Dim username, password

Dim rowNum

' Open Excel file

Set xlApp = CreateObject("Excel.Application")

xlApp.Visible = False ' Keep Excel hidden while running

Set xlBook = xlApp.Workbooks.Open("C:\path\to\your\LoginData.xlsx") ' Provide the full path to the Excel file

Set xlSheet = xlBook.Sheets(1)

' Find the number of rows in the sheet

rowNum = xlSheet.UsedRange.Rows.Count

' Loop through each row in the Excel file

For i = 2 To rowNum ' Assuming first row contains headers

' Read Username and Password from Excel

username = xlSheet.Cells(i, 1).Value

password = xlSheet.Cells(i, 2).Value

' Open the web browser and navigate to the login page

SystemUtil.Run "iexplore.exe", "http://example.com/login"

Browser("title:=Login").Page("title:=Login").Sync

' Enter the username and password

Browser("title:=Login").Page("title:=Login").WebEdit("username\_field").Set username

Browser("title:=Login").Page("title:=Login").WebEdit("password\_field").Set password

' Click the Login button

Browser("title:=Login").Page("title:=Login").WebButton("Login").Click

' Wait for the page to load (modify as needed)

Browser("title:=Dashboard").Page("title:=Dashboard").Sync

' Add any validation or verification steps here

' Example: Check if login was successful by looking for a specific element

If Browser("title:=Dashboard").Page("title:=Dashboard").WebElement("WelcomeMessage").Exist Then

Reporter.ReportEvent micPass, "Login Success", "Login was successful for user: " & username

Else

Reporter.ReportEvent micFail, "Login Failed", "Login failed for user: " & username

End If

' Close the browser after each iteration

Browser("title:=Dashboard").Close

Next

' Close Excel

xlBook.Close

xlApp.Quit

Set xlSheet = Nothing

Set xlBook = Nothing

Set xlApp = Nothing

**Explanation of the Script:**

1. **Excel Interaction**:
   * The script creates an instance of Excel and opens the **LoginData.xlsx** file.
   * It reads each row from the Excel sheet starting from the second row (to skip headers).
   * It stores the **username** and **password** from each row in variables.
2. **Login Test**:
   * For each username and password, the script opens the browser, navigates to the login page, and enters the login credentials.
   * It clicks the **Login** button and waits for the page to load.
   * You can add additional validation steps, such as checking for a **Welcome message** to verify if the login was successful.
3. **Results Reporting**:
   * After each login attempt, the script checks if a specific element (e.g., "WelcomeMessage") exists on the dashboard page to determine if the login was successful.
   * **Reporter.ReportEvent** is used to log the result in UFT’s test results.
4. **Loop**: The script loops through all the rows in the Excel file (except the first row containing headers) and performs the login for each username/password pair.
5. **Excel Cleanup**: After completing the loop, the script closes the Excel file and quits the Excel application to free resources.

**Steps to Run the Test:**

1. **Prepare your Excel file** (LoginData.xlsx) with the required login data.
2. **Update the file path** in the script to point to the correct location of your Excel file.
3. **Modify the web elements** (like username\_field, password\_field, Login, and Dashboard) to match the actual object names in your application.
4. **Run the test** by clicking the **Run** button in UFT.

**Q15: What are the different types of parameters available in UFT (e.g., test, action, and data table parameters)? Explain their use with examples.**

In UFT (Unified Functional Testing), parameters are used to pass dynamic values into test scripts, making the tests reusable and flexible. There are different types of parameters available in UFT, each serving specific purposes. These include **Test Parameters**, **Action Parameters**, and **Data Table Parameters**.

**1. Test Parameters:**

**Test parameters** are used to pass values from the test settings (Test level) to the actions within the test. These parameters allow for the customization of a test's execution without modifying the test script itself.

* **Purpose**: Test parameters are useful for passing values that are used across multiple actions in a test, such as environment configurations or login credentials.
* **Scope**: They are global and can be accessed by any action within the test.

**Example**: In UFT, you can set a test parameter for a login test with a username and password:

1. Go to **Test Settings** > **Parameters** and add parameters like Username and Password.
2. In the script, you can refer to these parameters:

' Get the test parameter values

Username = TestData("Username")

Password = TestData("Password")

' Use them in the login form

Browser.Page("micClass:=Page").WebEdit("name:=username").Set Username

Browser.Page("micClass:=Page").WebEdit("name:=password").Set Password

Browser.Page("micClass:=Page").WebButton("name:=login").Click

In this example, TestData("Username") fetches the value of the Username parameter defined in the Test Settings.

**2. Action Parameters:**

**Action parameters** are used to pass values to and from individual actions within a test. These are defined at the action level and are used when you need to pass values between actions, especially when the same action is reused in multiple places within the test.

* **Purpose**: Action parameters are useful for creating modular, reusable actions. For example, a "Login" action can accept different username/password combinations as parameters.
* **Scope**: The scope of action parameters is limited to the action in which they are defined.

**Example**:

1. Define an action parameter in an action's **Action Parameters** tab (e.g., Username, Password).
2. In the script, you can refer to these action parameters:

' Define action parameters

' Action 1 receives Username and Password parameters

' Use parameters in the script

Browser.Page("micClass:=Page").WebEdit("name:=username").Set Username

Browser.Page("micClass:=Page").WebEdit("name:=password").Set Password

Browser.Page("micClass:=Page").WebButton("name:=login").Click

In this example, Username and Password are passed to the action dynamically, allowing the action to be reused with different values.

**3. Data Table Parameters:**

**Data Table parameters** are used to pass values from the **Data Table** in UFT (a spreadsheet-like interface) to the test or actions. This is particularly useful in data-driven testing, where you need to execute the same test with different sets of data.

* **Purpose**: Data Table parameters allow you to run a test with multiple sets of input data, which is useful for validating the application under different conditions (e.g., different users, product names, etc.).
* **Scope**: The scope of Data Table parameters is global to the test, but they are primarily used for data-driven testing.

**Example**: In UFT, you can create a Data Table (like an Excel file) with columns for the test data (e.g., Username, Password):

| **Username** | **Password** |
| --- | --- |
| user1 | pass1 |
| user2 | pass2 |
| user3 | pass3 |

In your test script, you would reference the Data Table using the DataTable.Value method:

' Loop through the data table and get values for each iteration

For i = 1 To DataTable.GetRowCount

' Retrieve values from the Data Table

Username = DataTable.Value("Username", i)

Password = DataTable.Value("Password", i)

' Use them in the login form

Browser.Page("micClass:=Page").WebEdit("name:=username").Set Username

Browser.Page("micClass:=Page").WebEdit("name:=password").Set Password

Browser.Page("micClass:=Page").WebButton("name:=login").Click

Next

Here, the test runs multiple iterations, each time using a different set of login credentials from the Data Table

**6. Actions and Function Libraries:**

**Q16: What is an action in UFT? How does it help in organizing your test scripts? Create an example of a reusable action for logging into a web application.**

In Unified Functional Testing (UFT), an action is a reusable and modular component that represents a specific step or sequence of operations within a test script. Actions help in breaking down complex test scripts into smaller, manageable, and reusable units. By organizing tests into actions, you can enhance readability, maintainability, and reusability across multiple tests.

Benefits of using actions:

Modularity: Actions allow you to break down large tests into smaller parts, making the script easier to manage.

Reusability: Once an action is created, it can be reused across different test scripts, reducing the need for redundancy.

Maintainability: When changes are needed (e.g., updating a login process), you only need to modify the action, not every script that uses it.

Example of a reusable action for logging into a web application:

1. Create a new action in UFT, named LoginAction.
2. Define the steps required for logging in:
   * Open the browser and navigate to the login page.
   * Enter the username and password.
   * Click the login button.
3. Script for LoginAction:

' LoginAction - Logs into the web application

Browser("YourApp").Navigate "https://yourapplication.com/login"

Browser("YourApp").Page("Login").WebEdit("username").Set "testuser"

Browser("YourApp").Page("Login").WebEdit("password").Set "password123"

Browser("YourApp").Page("Login").WebButton("Login").Click

1. Call the action in the main test script:

MainTest - Uses LoginAction Call LoginAction

**Q17: Explain the concept of "Function Libraries" in UFT. How do you create and associate a function library with your test?**

In Unified Functional Testing (UFT), a "Function Library" is a collection of reusable functions that can be shared across multiple tests. These libraries help automate repetitive tasks and simplify test maintenance by centralizing commonly used functions in one place.

To create and associate a function library with your test:

1. Create a Function Library: In UFT, go to the "File" menu and select "New" > "Function Library."

Write the functions that will be used in your test. Functions in a library can perform actions like clicking buttons, verifying values, or other reusable operations.

Save the function library with a .vbs (VBScript) extension.

1. Associate the Function Library with Your Test:

Open your test in UFT.

From the "Test" menu, select "Settings" and then go to the "Libraries" tab.

Click on "Add" to browse and select the function library file (.vbs) you want to associate with your test.

Once added, the functions within the library can be called within the test scripts.

By using function libraries, you reduce code duplication and improve the maintainability of your tests.

**Q18: Write a simple function in a UFT function library that accepts two numbers as inputs and returns their sum. Call this function from your test script.**

1. Create the Function in the Function Library:

First, create a new function library in UFT.

Function Library Code:

' Function to calculate the sum of two numbers

Function AddNumbers(num1, num2)

' Return the sum of the two input numbers

AddNumbers = num1 + num2

End Function

Explanation:

* The function AddNumbers accepts two inputs: num1 and num2.
* It returns the sum of the two numbers using the AddNumbers = num1 + num2 statement.

2. Save the Function Library:

* Save this code in a Function Library file, e.g., MathFunctions.vbs.

3. Call the Function from Your Test Script:

Now, create a test script that calls this function and displays the result.

Test Script:

' Load the function library that contains the AddNumbers function

' Make sure to provide the full path to your function library file

' Example: "C:\Path\To\FunctionLibrary\MathFunctions.vbs"

FunctionLibrary "C:\Path\To\MathFunctions.vbs"

' Declare the numbers to be added

Dim num1, num2, result

' Assign values to num1 and num2

num1 = 5

num2 = 10

' Call the AddNumbers function and store the result

result = AddNumbers(num1, num2)

' Display the result in the Test Results window

Reporter.ReportEvent micInfo, "Sum Result", "The sum of " & num1 & " and " & num2 & " is " & result

Explanation:

* The FunctionLibrary statement is used to include the function library (MathFunctions.vbs) in your test script.
* num1 and num2 are the two numbers you want to add.
* The AddNumbers function is called with num1 and num2 as inputs, and the result is stored in the result variable.
* The Reporter.ReportEvent method is used to log the result in the UFT Test Results window.

4. Run the Test:

1. Open UFT and create a new Test.
2. Save the test.
3. Add the function library to the test (either through the Function Library settings or in the script as shown above).
4. Run the test by clicking Run or pressing F5.

Output:

* The script will calculate the sum of 5 and 10, and the result will be displayed in the Test Results window as:

The sum of 5 and 10 is 15 This simple example demonstrates how to create a function in a UFT function library and call it from a test script.

### **7. Descriptive Programming:**

**Q19: What is Descriptive Programming in UFT, and when would you use it? Write a UFT script using descriptive programming to click a button on a webpage (e.g., a "Submit" button).**

**Descriptive Programming in UFT**

**Descriptive Programming** in UFT (Unified Functional Testing) is a technique used to identify objects and perform actions without relying on the Object Repository. In descriptive programming, you provide the properties of the object in the script itself rather than storing those properties in the Object Repository. This approach allows for more dynamic, flexible, and efficient scripting, especially when dealing with objects that are not part of the Object Repository or have dynamic attributes.

**When to Use Descriptive Programming**

* **Dynamic Objects**: When objects have dynamically changing properties such as changing IDs or names, which can make object recognition through the Object Repository challenging.
* **Objects Not in Repository**: When you have a large number of objects that are not feasible to store in the Object Repository or when the application changes frequently.
* **Object Identification**: When you want to manipulate objects without adding them to the Object Repository, such as in situations where you have complex or temporary objects.

**UFT Script Using Descriptive Programming**

Suppose you need to click on a "Submit" button on a webpage using descriptive programming. Here's how you can do it:

' Create an object description for the Submit button

Dim SubmitButton

Set SubmitButton = Description.Create()

' Set properties of the Submit button

SubmitButton("html tag").Value = "INPUT" ' Tag of the button (can be INPUT or BUTTON depending on the type)

SubmitButton("type").Value = "submit" ' Type of the button (submit type)

SubmitButton("name").Value = "submitButton" ' Name of the button (it can be name, id, etc.)

' Use the description to identify the Submit button and click it

Browser("micClass:=Browser").Page("micClass:=Page").ChildObjects(SubmitButton)(0).Click

**Explanation:**

1. **Create Object Description**: Description.Create() is used to create a description object that will store the properties of the "Submit" button.
2. **Set Object Properties**: Using SubmitButton("property").Value, you define the properties that will be used to identify the button. In this example, we use html tag, type, and name. These can vary depending on the actual properties of the object you are interacting with.
3. **Identify and Click the Button**: The ChildObjects(SubmitButton) method is used to identify the button based on the properties defined in the SubmitButton object. (0) refers to the first matching object (if there are multiple matching elements). The Click method is then used to simulate the click on the button.

**When to Use This Script:**

* **Dynamic Web Elements**: If the button's properties (e.g., name or id) are likely to change dynamically or if you're working with a large application where many such objects exist.
* **No Object Repository**: If you do not want to store the object in the Object Repository and prefer writing a compact, reusable script.

**Q20: Explain the syntax for Descriptive Programming in UFT. Write a script that uses descriptive programming to interact with a web element based on its properties (e.g., link text, tagname, etc.).**

**Descriptive Programming Syntax in UFT**

Descriptive programming (DP) in UFT (Unified Functional Testing) allows you to interact with web elements dynamically by directly specifying their properties in the script. It eliminates the need to store objects in the Object Repository. The syntax for Descriptive Programming involves the following steps:

1. **Creating an Object Description**: You need to create a Description object that holds the properties used to identify the desired object.
2. **Assigning Properties**: You assign the properties of the object (like tagname, linktext, name, id, etc.) to the description object.
3. **Using the Description to Interact with the Object**: Use the description object to interact with the web element (like clicking on a button, entering text, etc.).

**Basic Syntax**

Dim objDescription

Set objDescription = Description.Create() ' Create a description object

' Set properties for the object

objDescription("property").Value = "value" ' Assign property-value pairs

' Use the object description to identify and interact with the object

Browser("micClass:=Browser").Page("micClass:=Page").ChildObjects(objDescription)(index).Action

**Example Script Using Descriptive Programming**

Let's write a script that interacts with a web element, such as a link on a webpage, based on its properties (e.g., link text and tag name).

**Scenario:**

You want to click on a link with the text "Learn More" on a webpage.

**Script:**

' Create an object description for the link

Dim linkDescription

Set linkDescription = Description.Create()

' Set the properties of the link

linkDescription("html tag").Value = "A" ' Tagname for the link (Anchor tag)

linkDescription("innertext").Value = "Learn More" ' The text of the link

' Use the description to identify and click the link

Browser("micClass:=Browser").Page("micClass:=Page").ChildObjects(linkDescription)(0).Click

**Explanation of the Script:**

1. **Create Object Description**:
   * Set linkDescription = Description.Create() creates a description object that will hold the properties for the web element (in this case, a link).
2. **Assign Properties**:
   * linkDescription("html tag").Value = "A" specifies that the element is an anchor (<A>) tag, which is used for links in HTML.
   * linkDescription("innertext").Value = "Learn More" specifies the text inside the link, which is "Learn More". This property helps in identifying the link based on its visible text.
3. **Identify and Click the Link**:
   * Browser("micClass:=Browser").Page("micClass:=Page") specifies the browser and page.
   * .ChildObjects(linkDescription) fetches all objects that match the description provided. (0) refers to the first match (in case there are multiple links with the same text).
   * .Click performs the click action on the identified link.

**Other Properties You Can Use in Descriptive Programming:**

**id**: Used to identify elements by their id attribute.

**name**: Used to identify elements by their name attribute.

**class**: Identifies elements by their class attribute.

**tagname**: Identifies elements by the type of HTML tag (e.g., A, INPUT, BUTTON).

**innertext**: Identifies elements by the text within the element (useful for links or buttons).

**value**: Useful for form elements like input fields.

**Q21: How does UFT handle dynamic objects with Descriptive Programming? Provide an example using a dynamic link or button.**

**Handling Dynamic Objects with Descriptive Programming in UFT**

In UFT (Unified Functional Testing), **dynamic objects** are those whose properties, such as id, name, text, or class, change frequently or are generated dynamically based on user interactions or session variables. **Descriptive Programming (DP)** allows you to handle such dynamic objects effectively by specifying their properties in the script itself, rather than relying on the Object Repository.

Using Descriptive Programming, you can identify dynamic objects based on the properties that may change from test run to test run, such as text content, id attributes, or even CSS classes that are dynamically assigned.

**How UFT Handles Dynamic Objects with Descriptive Programming:**

**Object Description**: UFT allows you to describe an object using its attributes (e.g., tagname, name, text, innertext, class, etc.). When dealing with dynamic objects, you can specify the properties that are most stable and unique to the object.

**Partial Matching**: UFT supports partial matching for properties. This is useful for objects whose attributes change, but the values follow a consistent pattern. For example, a button might have an id like submitButton\_12345, where the number 12345 is dynamic, but the prefix (submitButton\_) remains the same.

**Using Descriptive Programming for Dynamic Objects**: You can create a description object that only includes the stable, unique part of the dynamic object’s properties.

**Example of Descriptive Programming with Dynamic Button or Link:**

**Script to Handle a Dynamic Link:**

' Create a description object for the dynamic link

Dim linkDescription

Set linkDescription = Description.Create()

' Set the properties of the link

linkDescription("html tag").Value = "A" ' Anchor tag

linkDescription("innertext").Value = "Submit" ' The static link text (e.g., "Submit")

' Use the description object to find and click the dynamic link

Browser("micClass:=Browser").Page("micClass:=Page").ChildObjects(linkDescription)(0).Click

**Explanation of the Script:**

**Creating Object Description**:

Set linkDescription = Description.Create() creates a description object that will hold the attributes used to identify the dynamic link.

**Assigning Properties**:

* + linkDescription("html tag").Value = "A" tells UFT that you are dealing with an anchor (<A>) tag, which is used for links.
  + linkDescription("innertext").Value = "Submit" specifies that you want to identify the link based on its text ("Submit"). Even if the id or name changes dynamically, the text content ("Submit") remains the same.

**Interacting with the Dynamic Object**:

* + ChildObjects(linkDescription)(0).Click is used to identify the first matching object that fits the description. The (0) refers to the first matching link on the page.
  + The .Click method is used to perform the action on the identified link.

**Example of Descriptive Programming with Dynamic Button:**

**Scenario:**

In this case, you have a dynamic button where the id attribute is generated dynamically, but the name or class attribute is static.

**Script to Handle a Dynamic Button:**

' Create a description object for the dynamic button

Dim buttonDescription

Set buttonDescription = Description.Create()

' Set the properties of the button

buttonDescription("html tag").Value = "BUTTON" ' Tagname for the button

buttonDescription("class").Value = "submit-button" ' Static class name of the button

' Use the description object to identify and click the dynamic button

Browser("micClass:=Browser").Page("micClass:=Page").ChildObjects(buttonDescription)(0).Click

**Explanation of the Script:**

1. **Creating Object Description**:Set buttonDescription = Description.Create() creates a description object for the button.
2. **Assigning Properties**:
   * buttonDescription("html tag").Value = "BUTTON" identifies the element as a button.
   * buttonDescription("class").Value = "submit-button" targets the button by its static class attribute, which does not change dynamically (while the id might).
3. **Interacting with the Dynamic Object**:ChildObjects(buttonDescription)(0).Click identifies the first matching button based on the description and clicks it.

### **8. Synchronization and Wait Statements:**

**Q22: Why is synchronization important in UFT? What are the different synchronization techniques you can use to make sure your script waits for an element to be available?**

**Why is Synchronization Important in UFT?**

Synchronization is crucial in **UFT** because it ensures that your test scripts interact with elements only when they are fully loaded and ready. Without proper synchronization, UFT might attempt to interact with an element before it becomes available, leading to **false failures**, **script crashes**, or **incorrect results**. It ensures the test runs reliably, even when dealing with slow-loading pages, dynamic content, or asynchronous elements.

**Synchronization Techniques in UFT:**

Here are the different synchronization techniques you can use in UFT to ensure your script waits for an element to be available before interacting with it:

**1. Sync Method:**

**Purpose**: Makes UFT wait until the page or application is fully loaded before performing actions.

**When to use**: Use when waiting for the entire page or application to load.

**Example**: Browser("MyBrowser").Page("MyPage").Sync

**2. Wait Method:**

**Purpose**: Pauses the execution for a specified amount of time (in seconds). It is useful for waiting for elements to load.

**When to use**: Use when you know how long an element will take to load, or if the loading time is predictable.

**Example**: Browser("MyBrowser").Page("MyPage").WebButton("Submit").Wait(10) ' Waits for 10 seconds

**3. Exist Method:**

**Purpose**: Checks if an element is present on the page. Returns True if the element exists, False otherwise.

**When to use**: Use when you need to verify that an element is available before interacting with it.

**Example**: If Browser("MyBrowser").Page("MyPage").WebButton("Submit").Exist(10) Then

Browser("MyBrowser").Page("MyPage").WebButton("Submit").Click

Else

Reporter.ReportEvent micFail, "Submit button not found", "The Submit button is missing or took too long to appear"

End If

**4. WaitProperty Method:**

**Purpose**: Waits for a specific property of an object to meet a condition (e.g., waiting for an element to become visible).

**When to use**: Ideal for dynamic content (e.g., elements that load asynchronously) or when you need to wait for specific properties (like visibility, enabled state, etc.).

**Example**: Browser("MyBrowser").Page("MyPage").WebButton("Submit").WaitProperty "visible", True, 10 ' Waits for the Submit button to become visible

**5. Object Sync Method:**

**Purpose**: Synchronizes UFT with an object's state (e.g., it waits for an object to be ready for interaction).

**When to use**: Use when you want to synchronize with a specific object instead of waiting for the entire page.

**Example**: Browser("MyBrowser").Page("MyPage").WebButton("Submit").ObjectSync

**6. Do...Loop with Exist or WaitProperty:**

**Purpose**: Repeatedly checks for the existence or property of an element in a loop until the condition is met or a timeout occurs.

**When to use**: Use when you want to retry waiting for an element multiple times before giving up.

**Example**: Dim retryCount retryCount = 0

Do While Browser("MyBrowser").Page("MyPage").WebElement("DynamicContent").Exist(5) = False And retryCount < 3

retryCount = retryCount + 1

Wait(5)

Loop

If retryCount = 3 Then

Reporter.ReportEvent micFail, "Dynamic Content Not Loaded", "The dynamic content took too long to load"

End If

**7. Timeout Property:**

**Purpose**: Adjusts the timeout for object synchronization (i.e., how long UFT should wait for an object to be available before proceeding).

**When to use**: Use to give objects more time to load if they are taking longer than the default timeout.

**Example**: Browser("MyBrowser").Page("MyPage").WebButton("Submit").Timeout = 20 ' Set timeout to 20 seconds

**Q23: Write a script that uses the Sync method and Wait method to ensure UFT waits for a page to load before performing actions like clicking a button.**

UFT script that uses the **Sync** and **Wait** methods to ensure UFT waits for a page to load before clicking a button:

' Start the test case

' Navigate to the page (replace with actual URL and browser)

Browser("MyBrowser").Navigate "https://example.com"

' Use Sync method to wait for the page to load completely

Browser("MyBrowser").Page("MyPage").Sync

' Alternatively, use Wait method to wait for a specific element to load

Browser("MyBrowser").Page("MyPage").WebButton("Submit").Wait(10) ' Waits for up to 10 seconds for the Submit button

' Check if the button exists before clicking it

If Browser("MyBrowser").Page("MyPage").WebButton("Submit").Exist Then

' Click the button after ensuring the page is ready

Browser("MyBrowser").Page("MyPage").WebButton("Submit").Click

Else

Reporter.ReportEvent micFail, "Button not found", "The Submit button was not found on the page."

End If

**Explanation:**

1. **Sync Method**:
   * Browser("MyBrowser").Page("MyPage").Sync: This ensures UFT waits until the page is fully loaded before proceeding with any actions.
2. **Wait Method**:
   * Browser("MyBrowser").Page("MyPage").WebButton("Submit").Wait(10): This waits for up to 10 seconds for the "Submit" button to be visible and interactive. If the button doesn’t appear in that time, UFT will proceed without interacting with it.
3. **Check Object Existence**:
   * If Browser("MyBrowser").Page("MyPage").WebButton("Submit").Exist: This checks if the button exists before clicking it, preventing errors if the element is not yet loaded or available.

**Q24: How would you handle synchronization issues when testing a slow application or a page with dynamic content?**

**Handling Synchronization Issues:**

Synchronization issues occur when UFT tries to interact with elements before they are ready (due to slow page loads or dynamic content). To handle this, you can use various synchronization techniques in UFT.

**1. Implicit Waits:**

**Use the Sync method** to tell UFT to wait until an object is fully loaded and ready for interaction.

**Example**:Browser("MyBrowser").Page("MyPage").Sync

This will wait until the page is ready before performing further actions.

**2. Explicit Waits (Wait/WaitProperty):**

**Wait method**: Forces UFT to wait for a specified amount of time before executing the next step.

**Example**:Browser("MyBrowser").Page("MyPage").WebButton("Submit").Wait(10) ' Waits for 10 seconds

**WaitProperty method**: Waits for a specific property of an object to change (ideal for dynamic content).

**Example**:Browser("MyBrowser").Page("MyPage").WebButton("Submit").WaitProperty "visible", True, 10

This waits for the "Submit" button to become visible before interacting with it, within 10 seconds.

**3. Use of Exist Method:**

**Check if an object exists** before interacting with it. This is useful for dynamic content that may appear after a delay.

**Example**:If Browser("MyBrowser").Page("MyPage").WebButton("Submit").Exist(10) Then

Browser("MyBrowser").Page("MyPage").WebButton("Submit").Click

Else

Reporter.ReportEvent micFail, "Submit button not found", "The button is missing or took too long to appear"

End If

**4. Use of Object Sync:**

**Object synchronization** can be used to synchronize objects, such as buttons or links, ensuring that they are ready for interaction before performing actions.

**Example**:Browser("MyBrowser").Page("MyPage").WebButton("Submit").ObjectSync

**5. Dynamic Content Handling:**

For dynamic content, such as elements that load asynchronously (e.g., AJAX), use **Exist** or **WaitProperty** to synchronize specific elements.

You can also use **Do...Loop** to repeatedly check for an element until it's available or until a timeout occurs.

**Example**: Dim retryCount

retryCount = 0

Do While Browser("MyBrowser").Page("MyPage").WebElement("DynamicContent").Exist(5) = False And retryCount < 3

retryCount = retryCount + 1

Wait(5)

Loop

If retryCount = 3 Then

Reporter.ReportEvent micFail, "Dynamic Content Not Loaded", "The dynamic content took too long to load"

End If

**6. Use Timeout Settings:**

For actions that may take time due to slow loading, increase the **timeout** value for object identification.

**Example**:Browser("MyBrowser").Page("MyPage").WebButton("Submit").Timeout = 20 ' Set timeout to 20 seconds

### **9. Error Handling and Recovery:**

**Q25: How can you add exception handling in UFT to handle pop-ups or alerts that appear unexpectedly during the test execution?**

**Adding Exception Handling in UFT for Pop-ups/Alerts:**

You can use the Recovery Scenario or custom VBScript error handling to manage unexpected pop-ups or alerts in UFT.

**1. Using Recovery Scenarios:**

**Recovery Scenarios** in UFT are pre-configured actions that are automatically triggered when specific conditions (like pop-ups or alerts) occur during test execution.

**Steps**:

**Create a Recovery Scenario**:

* + - Go to Tools > Recovery Scenario Manager.
    - Click on **New** and define the condition (e.g., an unexpected pop-up or alert).
    - Specify the action you want UFT to take (e.g., click "OK", close the alert, etc.).

**Associate the Recovery Scenario**:

* + - Once created, associate the Recovery Scenario with your test by selecting Test Settings > Recovery.
    - Choose the relevant recovery scenario for your test.

**Automatic Execution**:

* + - When a pop-up or alert appears, UFT will automatically trigger the defined recovery action based on the scenario.

**2. Using VBScript for Custom Error Handling:**

For more control, you can add **custom error handling** using On Error Resume Next and If...Then checks in your test scripts.

**Example for handling alerts:**

' Enable error handling

On Error Resume Next

' Attempt to interact with a button that may cause a pop-up

Browser("MyBrowser").Page("MyPage").WebButton("Submit").Click

' Check for an alert or pop-up

If Err.Number <> 0 Then

' If an error occurs, check if it's an alert or pop-up

If Browser("MyBrowser").Dialog("Alert").Exist Then

' Handle the pop-up (e.g., click "OK")

Browser("MyBrowser").Dialog("Alert").WinButton("OK").Click

MsgBox "Pop-up handled successfully!"

End If

' Clear the error

Err.Clear

End If

' Continue with other steps

On Error GoTo 0

**How It Works:**

**On Error Resume Next**: Tells UFT to continue execution even if an error occurs.

**Err.Number**: Checks if an error (e.g., a pop-up) was triggered.

**Browser().Dialog().Exist**: Verifies if a pop-up or alert is present.

**Err.Clear**: Clears the error after handling it.

### **10. Test Results and Reporting:**

**Q26: Explain how UFT generates test results. How do you view and analyze the test results after running a test in UFT?**

**How UFT Generates Test Results:**

**Execution Log**: During test execution, UFT logs the status (pass/fail) of each step, along with any messages, screenshots, or errors that occur.

**Reporter Object**: UFT uses the Reporter object to capture the status of each test step. By default, UFT generates a detailed report for each test run, including step descriptions, results, and messages.

**Test Results File**: UFT saves the results in a structured format (usually in the form of an HTML file) that contains detailed information about the test steps, status, and any failures or errors.

**Viewing and Analyzing Test Results in UFT:**

1. **View Results in the "Test Results" Tab**:
   * After running a test, go to the **"Test Results" tab** in UFT.
   * This tab displays the overall execution results, including pass/fail status, step descriptions, and any associated messages or snapshots.
2. **Examine Detailed Test Results**:
   * Click on individual steps to view more detailed information like error messages, failure reasons, or snapshots captured during the test run.
   * UFT provides a **log** of what occurred during each step, including any failures or unexpected behaviors.
3. **Generate a Test Results Report**:
   * UFT automatically generates an HTML report, which can be found in the output folder or the directory you specified for results.
   * Open this report in a web browser to view a detailed, formatted summary of the test execution, with timestamps, status, and any error messages.
4. **Analyze Errors and Failures**:
   * If a test fails, look at the **failure reason** and associated **screenshot** (if configured) to understand what went wrong.
   * Check if the error is due to object identification issues, incorrect test data, or script logic errors.
   * Use the **Run-Time Data Table** to see the values fed into the test, and verify if incorrect data led to the failure.

**Q27: What is the difference between the "Test Results" tab and the "Run-Time Data Table" in UFT? How would you use them to debug a failing test?**

**Test Results Tab:**

**Purpose**: Displays the overall execution status of the test, including passed, failed, or skipped steps.

**Contents**: Shows details of each test step (e.g., pass/fail status, messages, snapshots).

**Usage for Debugging**:

* + **Identify Failed Steps**: Quickly spot where the test failed by looking at the status for each step.
  + **Messages**: Check the failure messages and snapshots (if enabled) for clues on what went wrong.
  + **Execution Logs**: Provides detailed logs of actions taken during the test execution.

**Run-Time Data Table:**

**Purpose**: Stores data used during test execution, such as input values, output values, and envirnment variables.

**Contents**: Displays the values that were available at runtime for the data-driven tests.

**Usage for Debugging**:

* + **Check Test Data**: Verify if the data fed into the test was correct.
  + **Check Data Flow**: Track how data changes during execution and whether it affects the test outcome.
  + **Examine Variable Values**: Look at the variable states during each iteration (for data-driven tests) to identify incorrect or missing data.

**How to Use Them to Debug a Failing Test:**

**Check the Test Results Tab**:

* + Look for any failed steps and review the associated messages or snapshots for errors.
  + Understand which part of the test failed (e.g., an action, validation, or object identification).

**Inspect the Run-Time Data Table**:

* + Verify if the test data is correct and if any variables are missing or incorrect.
  + If using a data-driven approach, check the values used for each iteration to confirm whether bad data caused the failure.

**Q28: Write a script that generates a custom report in UFT after executing a test case. This report should include test steps, status (pass/fail), and any relevant messages.**

' Start of the test case

Reporter.Filter = rfDisableAll

Reporter.ReporterStatus = False

' Example test case

Dim testStep, status, message

' Create custom report file

Dim reportFile

Set reportFile = CreateObject("Scripting.FileSystemObject").CreateTextFile("C:\TestReport.txt", True)

' Write header to report

reportFile.WriteLine "Test Report"

reportFile.WriteLine "====================="

' Test Step 1: Example step (replace with actual test steps)

testStep = "Open Application"

status = "Pass" ' Change based on actual status

message = "Application opened successfully"

reportFile.WriteLine "Step: " & testStep

reportFile.WriteLine "Status: " & status

reportFile.WriteLine "Message: " & message

reportFile.WriteLine "-----------------------------"

' Test Step 2: Another example step

testStep = "Click on Button"

status = "Fail" ' Change based on actual status

message = "Button click failed"

reportFile.WriteLine "Step: " & testStep

reportFile.WriteLine "Status: " & status

reportFile.WriteLine "Message: " & message

reportFile.WriteLine "-----------------------------"

' Close report file

reportFile.Close

' Reset reporter status for further test cases

Reporter.ReporterStatus = True

Reporter.Filter = rfEnableAll