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?

  UFT (earlier called QTP) is a software made by Micro Focus that helps testers automate testing of software—especially when clicking buttons, filling forms, or checking results manually becomes boring and time-consuming. Now, how is UFT different?
- **UFT vs QTP:** Actually, UFT is the upgraded version of QTP. So it's more like "QTP 2.0" – it supports more technologies, better scripting, and even some mobile testing now.
- **UFT vs Selenium:** Selenium is open-source (free) and mainly used for web testing. UFT is a paid tool, but it can test not just web apps, but also desktop apps, SAP, Java, Oracle apps, etc. Selenium needs a lot of setup and coding, while UFT is easier to use with a record-and-playback feature and VBScript scripting.

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

  UFT comes loaded with handy features for testers. Here's a quick overview in a friendly way:

- **Record and Playback:**
  You click, type, and interact with the app—and UFT records everything and plays it back later like a video.
- **Object Repository:**
  It remembers all the buttons, fields, and other stuff it sees during the test, so you can reuse them in your scripts.
- **Smart Identification:**
  If something changes slightly (like a button moved), UFT still tries to recognize it instead of failing instantly.
- **Data-Driven Testing:**
  You can test the same thing with different inputs (like usernames, passwords, etc.) using Excel files easily.
- **Built-in Reports:**
  After running tests, it shows what passed, what failed, and screenshots too—so you don't need extra tools for that.

How it helps in testing:

- **Functional Testing:**
  You can check if your software is working as expected. For example: clicking a "Login" button should log you in.
- **Regression Testing:**
  After updates or bug fixes, you can re-run old test cases to make sure nothing else got broken.
- **GUI Testing:**
  It checks if the front-end (like buttons, text boxes, layouts) are working and looking fine across versions.

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

In UFT, anything you interact with in an app (like a button or a drop-down) is called an **object**. UFT is smart enough to spot many types of them.

1. **Standard GUI Objects** ○ These are basic things like buttons, text fields, checkboxes. ○ **Examples:**
   - A "Submit" button on a form
   - A text box where you enter your name

2. **Web Objects** ○ Used when testing web applications.
   - **Examples:**
     - Hyperlinks
     - Web buttons

3. **ActiveX / Java / .NET Objects** ○ For apps built using these techs. ○ **Examples:**
   - Java dropdown menu
   - .NET Grid control

4. **Custom Objects / Non-standard Controls** ○ These are weird or custom-made UI components not recognized normally.

UFT may need extra help here (like using insight or descriptive programming). ○ **Examples:**

- A canvas-based drawing tool
- A fancy slider or animation-based input

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.

```
'==================================
' Open Notepad, Type and Save File
'==================================

' Step 1: Launch Notepad
SystemUtil.Run "notepad.exe"

' Step 2: Wait for Notepad to appear
Window("nativeclass:=Notepad").WaitProperty "visible", True, 10

' Step 3: Type the message
Window("nativeclass:=Notepad").WinEditor("nativeclass:=Edit").Set "Hello, this is a UFT test!"

' Step 4: Save the file using File > Save As
Window("nativeclass:=Notepad").WinMenu("menuobjtype:=2").Select "File;Save As"

' Step 5: Set file name and save
Window("title:=Save As").WinEdit("attached text:=File name:").Set "C:\Users\Ananya\Desktop\UFTTest.txt"
Window("title:=Save As").WinButton("text:=Save").Click

' Step 6: Close Notepad
Window("nativeclass:=Notepad").Close

' Step 7: If Save confirmation appears (optional), click Don't Save
If Window("title:=Notepad").Dialog("text:=Notepad").WinButton("text:=Don't Save").Exist(2) Then
```

```
    Window("title:=Notepad").Dialog("text:=Notepad").WinButton("text:=Don't
Save").Click
End If

' Step 8: Report success
Reporter.ReportEvent micPass, "File Saved", "Text successfully saved to Notepad file."
```

- **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.

```
'============================
' Google Search using UFT
'============================

' Step 1: Launch the browser and navigate to Google
SystemUtil.Run "chrome.exe", "https://www.google.com"

' Step 2: Wait for the browser to load completely
Browser("micClass:=Browser").Sync

' Step 3: Enter search term into the input box
Browser("micClass:=Browser").Page("micClass:=Page").WebEdit("name:=q").Set "UFT
Automation"

' Optional wait (let suggestions load)
Wait 2

' Step 4: Click the Google Search button
' Since Google dynamically updates buttons, we use index or outertext property
Browser("micClass:=Browser").Page("micClass:=Page").WebButton("name:=btnK",
"index:=1").Click

' Step 5: Wait and Report
```

- Wait 3
- Reporter.ReportEvent micPass, "Search Executed", "Searched for 'UFT Automation' on Google"

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, when you're testing something (like clicking a button or typing in a textbox), UFT needs to know what that object is and how to identify it. For that, it saves information about those objects—like name, type, location, etc.—in something called an Object Repository.

  1. **Local Object Repository (LOR):**

- This is created automatically when you record actions in a test.
- It is attached to a specific test only.
- Think of it like keeping notes inside one notebook—you can't use it outside that notebook.
- **Location:** Stored within the test's folder.

  Example:

  If you record a test called LoginTest, the objects (like Username field, Login button) get saved in that test's local repo.

  2. **Shared Object Repository (SOR):**

- This is a separate file that stores objects and can be used in multiple tests.
- Great for big projects where multiple tests use the same objects.
- You create it manually using the Object Repository Manager and then link it to any test.
- Location: Saved as .tsr files (Test Shared Repository).

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

  When you're testing an application, UFT has to find and interact with things like buttons, fields, or dropdowns. That process of how UFT identifies and understands objects on the screen is called Object Identification.

  How does UFT identify objects?

  UFT looks at certain properties of objects to recognize them. For example:

- A button might be identified by:
  - html tag = button
  - name = Submit ○
    type = submit

UFT stores these properties in the Object Repository and uses them to find the object again when the test runs.

Types of Identification in UFT

1. **Standard Identification (Normal way):**
    - UFT uses a set of default properties like name, class, type, ID, etc.
    - If these match, it says: “Yes, I found it!

2. **Smart Identification (Backup Plan):**
    - Sometimes, objects might change (e.g., their position, ID).
    - If UFT can’t find the object using normal properties, it uses Smart Identification, which tries to find the "closest match" using other properties like XPath, layout, etc.

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

In UFT, a Checkpoint is like a verification step.
It checks whether something in your application is correct during test execution— for example, whether a label is displaying the right text, or whether a value in a table is correct.
In UFT, a Checkpoint is like a verification step.
It checks whether something in your application is correct during test execution— for example, whether a label is displaying the right text, or whether a value in a table is correct.

```
'=======================================
' Verify text on a web page using Text Checkpoint
'=======================================

' Step 1: Launch browser and navigate to a page
SystemUtil.Run "chrome.exe", "https://example.com"

' Step 2: Wait for page to load
Browser("micClass:=Browser").Sync
```

```
' Step 3: Add a Text Checkpoint dynamically using .CheckProperty Dim
expectedText
expectedText = "Welcome, Ananya!"

' Option A: Use WebElement to check text
If
Browser("micClass:=Browser").Page("micClass:=Page").WebElement("innertext:=" &
expectedText).Exist(5) Then
  Reporter.ReportEvent micPass, "Text Checkpoint", "Text '" & expectedText & "' is
present on the page."
Else
   Reporter.ReportEvent micFail, "Text Checkpoint", "Text '" & expectedText & "' is NOT
found on the page."
End If
```

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

1. Standard Checkpoint Definition:

A *Standard Checkpoint* in UFT is used to verify the properties of user interface (UI) elements in the application under test. It can check attributes such as text, enabled/disabled status, visibility, and more.

Use Case Example:

Suppose an e-commerce application displays a confirmation message “Order placed successfully” after the user completes a purchase. A Standard Checkpoint can be added to verify that this message is displayed correctly on the screen. **When to Use:**

- To validate UI elements like labels, buttons, links, images, and other objects present in the graphical user interface.
- Suitable for functional and GUI validation.

2. Database Checkpoint Definition:

A *Database Checkpoint* is used to validate data stored in a database. UFT can connect to a database using a connection string, execute SQL queries, and compare the returned data with expected results.

Use Case Example:

In a banking application, when a user transfers funds between accounts, the account balance should be updated accordingly in the backend database. A

Database Checkpoint can be used to run a query and verify that the updated balance is reflected in the relevant database table. **When to Use:**

- To validate data stored in backend databases such as Oracle, SQL Server, MySQL, etc.
- Useful in scenarios where data integrity and consistency need to be verified beyond the user interface.

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

  UFT provides several methods to handle such dynamic behavior:

  A. Descriptive Programming (DP)
  Descriptive Programming allows testers to define objects directly in the script, rather than relying on the Object Repository. This is useful for handling objects whose properties change frequently.
  Example: Handling a Button with Dynamic Text
  Suppose you have a button that changes text based on user interaction. Initially, it says "Start", and after clicking, it changes to "Stop".

  B. Using Regular Expressions in Object Repository
  You can also update the properties in the Object Repository to use regular expressions instead of fixed values. This allows UFT to match a range of possible values. **Steps:**

1. Open the Object Repository.
2. Select the dynamic object (e.g., a button).
3. Change the property value (e.g., text) to something like: Start.* or Stop.*.
4. Enable the "Regular Expression" checkbox.
   This way, UFT will match any button whose text starts with "Start" or "Stop".

  C. Smart Identification (Fallback Option)
  If UFT fails to uniquely identify the object due to dynamic changes, it can use Smart Identification, which applies a set of secondary properties to locate the object. However, Smart Identification should not be the first approach, as it increases execution time and may produce false positives.

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).

Parameterization in UFT refers to the process of replacing hardcoded values in test scripts with variables or parameters. This allows the test to run multiple times with different sets of input data without changing the code each time.

Parameterization makes test scripts:

- **Reusable** – run the same script with different input values
- **Efficient** – avoids duplication of similar test steps
- **Maintainable** – if input values change, update them in one place only
- **Scalable** – useful for data-driven testing (e.g., multiple users, products, or transactions)

In real-world testing, hardcoding inputs (e.g., usernames, passwords, search terms) is not practical. Parameterization helps simulate real scenarios.

3. Ways to Parameterize in UFT UFT allows parameterization using:

Method	Description
Data Table (Global/Local)	Built-in Excel-like sheet to provide test data
Environment Variables	Parameters defined outside the test
Test/Action Parameters	Parameters passed to actions
Random Numbers / Custom Code	Generated values at runtime

```
'==============================
' Parameterized Login Script
'==============================
```

```
' Open the login page
SystemUtil.Run "chrome.exe", "https://example.com/login"  ' Replace with actual
URL

' Loop through all rows in the DataTable
rowCount = DataTable.GetRowCount

For i = 1 To rowCount
   DataTable.SetCurrentRow(i)

   ' Get values from parameterized columns     username =
DataTable.Value("Username", dtGlobalSheet)     password =
DataTable.Value("Password", dtGlobalSheet)

   ' Fill login form
   Browser("micClass:=Browser").Page("micClass:=Page").WebEdit("html id:=user").Set
username
   Browser("micClass:=Browser").Page("micClass:=Page").WebEdit("html id:=pass").Set
password
   Browser("micClass:=Browser").Page("micClass:=Page").WebButton("html
id:=loginBtn").Click

   ' Log result
   Reporter.ReportEvent micDone, "Login", "Tested with username: " & username

   Wait 2 ' Small delay before next run
Next
```

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

```
'=========================================
' Data-Driven Login Test from Excel in UFT
'=========================================

' Load Excel file using DataTable.ImportExternal
DataTable.ImportSheet "C:\UFT_Data\LoginData.xlsx", 1, "Global"
```

```
' Get total rows (excluding header) Dim
rowCount
rowCount = DataTable.GetRowCount

' Launch browser
SystemUtil.Run "chrome.exe", "https://example.com/login"   ' Change to your actual
login URL

' Loop through each row of data
For i = 1 To rowCount
   DataTable.SetCurrentRow(i)

   ' Read username and password from Excel     user =
DataTable.Value("Username", dtGlobalSheet)     pass =
DataTable.Value("Password", dtGlobalSheet)

   ' Set credentials in form
   Browser("micClass:=Browser").Page("micClass:=Page").WebEdit("html id:=user").Set
user
   Browser("micClass:=Browser").Page("micClass:=Page").WebEdit("html id:=pass").Set
pass
   Browser("micClass:=Browser").Page("micClass:=Page").WebButton("html
id:=loginBtn").Click

   ' Log step
   Reporter.ReportEvent micPass, "Login Attempt", "Tried with Username: " & user & "
and Password: " & pass

   ' Add wait and logout if needed (optional)
   Wait 2
Next
```

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

Here are the main types of parameters available in UFT:

1. **Test Parameters Definition:**

Test parameters are used to pass values from outside the test (usually during test execution) into the test. **How to Create:**

- Go to File → Settings → Parameters → Test Parameters
- Add a parameter, e.g., EnvironmentURL = http://testsite.com **Purpose:**
- Pass environment-specific, user-specific, or configuration-specific values.
- Often used in batch test runs or test management tools (like ALM).

2. Action Parameters

Definition:

Action parameters allow you to pass values between actions in a test. They help in making actions more reusable and modular. **Types:**

- **Input Parameters:** Passed into the action
- **Output Parameters:** Returned from the action **How to Create:**
- Right-click on the action > Action Properties
- Add Username and Password as input parameters **Purpose:**
- Promotes reusability of actions
- Helps in **data passing** between multiple actions

3. Data Table Parameters

Definition:

Data Table parameters are used for data-driven testing, where values come from UFT's built-in spreadsheet-like Data Table (Global or Action sheets).

How to Use:

- Go to View → Data Table
- Add columns: Username, Password
- Fill in multiple rows of data **Purpose:**
- Supports repeated execution of test cases with varying data
- Useful for functional testing with multiple scenarios

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. An Action in UFT is a modular block of test steps that can be:
- Created once and reused multiple times
- Called from within the same test or from other tests Actions help in:
- Organizing your script into logical parts (e.g., Login, Search, Logout)

- Promoting reusability
- Making tests easier to read and maintain

How It Helps in Organizing Scripts:

- Breaks a large test into **smaller manageable blocks**
- Encourages **code reuse** (no duplication)
- Makes test maintenance easier: fix in one place, reflect everywhere

A) Create the Action:

1. In your UFT Test, go to Keyword View or Action menu.
2. Click: Insert > Call to New Action
3. Name it: LoginAction
4. In Action Properties, set Reusable Action = True

B) Add This Code Inside the Action:

```
' LoginAction - Reusable login action

Browser("micClass:=Browser").Page("micClass:=Page").WebEdit("html id:=user").Set
"admin" Browser("micClass:=Browser").Page("micClass:=Page").WebEdit("html
id:=pass").Set "admin123"
Browser("micClass:=Browser").Page("micClass:=Page").WebButton("html
id:=loginBtn").Click

Reporter.ReportEvent micPass, "Login", "User successfully logged in."
```

C) Call This Action from Another Test: In any other test:

1. Go to Insert > Call to Existing Action
2. Browse to the test where LoginAction is saved
3. Select it and it will appear in your test flow

- **Q17:** Explain the concept of "Function Libraries" in UFT. How do you create and associate a function library with your test?
- A Function Library in UFT is a separate file that contains user-defined functions, procedures, and reusable code blocks that can be called from one or more tests. These libraries help in keeping the test scripts modular, clean, and maintainable.
- UFT function libraries are typically saved with the .vbs (VBScript) or .qfl file extension.

Steps to Create and Associate a Function Library in UFT

Step 1: Create the Function Library

- Open UFT
- Go to **File → New → Function Library**
- Write your reusable functions using VBScript
- Save the file with a .vbs or .qfl extension
  (e.g., LoginFunctions.vbs)

Step 2: Associate the Library with Your Test

There are two ways to associate a function library:

Method A: Using GUI (Recommended)

1. Open your test
2. Go to File → Settings → Resources (tab)
3. Click on the **"+"** button
4. Browse and select your .vbs or .qfl file
5. Click OK

Method B: Associating in Code

You can also associate the function library using the ExecuteFile statement in test script:

```
ExecuteFile "C:\UFT\Libraries\LoginFunctions.vbs"
```

- **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.

```
'-------------------------------- '
File: MathLibrary.vbs
' Description: Contains a simple addition function
'--------------------------------

Function AddNumbers(num1, num2)
   AddNumbers = num1 + num2 End
Function
```

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 (DP) in UFT is a method where you define objects directly in the script**,** instead of storing them in the Object Repository.
  This approach allows UFT to interact with application objects at runtime using their properties, without relying on pre-recorded objects.

  '==========

  ' Click "Submit" using Descriptive Programming

  '==========

  ' Step 1: Launch the browser and navigate to a webpage

  SystemUtil.Run "chrome.exe", "https://example.com"
  ' Step 2: Sync browser (wait for page to load)

  Browser("micClass:=Browser").Sync

  ' Step 3: Define the "Submit" button using description

  Dim oButton

  Set oButton = Description.Create oButton("micClass").Value

  = "WebButton" oButton("html tag").Value = "INPUT"

  oButton("type").Value = "submit"

  oButton("name").Value = "submit" ' OR you can use "value" := "Submit" depending on actual property

  ' Step 4: Click the button

  Browser("micClass:=Browser").Page("micClass:=Page").WebButton(oButton).Click

  ' Step 5: Log result

  Reporter.ReportEvent micPass, "Submit Button", "Submit button clicked successfully."

- **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 (DP) allows UFT to interact with objects directly using their properties in the script, without depending on the Object Repository. There are two types of syntax for DP:

A. Static Descriptive Programming Syntax
This method embeds the object's identifying properties directly within the test step using "property:=value" pairs.
Syntax:
ObjectType("property1:=value1", "property2:=value2").Operation **Example:**
Browser("name:=MyApp").Page("title:=Home").
Link("text:=Contact Us", "html tag:=A").Click

B. Dynamic Descriptive Programming (Using Description Object)
This method uses a Description object to define object properties dynamically.
Syntax:
Set objDesc = Description.Create() objDesc("property1").Value =
"value1" objDesc("property2").Value = "value2"
...
ObjectType(objDesc).Operation

Script

```
' Step 1: Launch browser

SystemUtil.Run "chrome.exe", "https://example.com"


' Step 2: Wait for page load

Browser("micClass:=Browser").Page("micClass:=Page").Sync


' Step 3: Define the link using descriptive programming

Set linkDesc = Description.Create

linkDesc("micClass").Value = "Link"
```

```
linkDesc("innertext").Value = "About Us"

linkDesc("html tag").Value = "A"


' Step 4: Click the link

Browser("micClass:=Browser").Page("micClass:=Page").Link(linkDesc).Click


' Step 5: Report

Reporter.ReportEvent micPass, "Descriptive Click", "Successfully clicked the 'About Us' link."
```

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

In web or desktop applications, some UI objects (like links, buttons, or text fields) have dynamic properties that change every time the application runs—for example:

- A button's name might include a session ID (btn_12345)
- A link's text might change based on user data (Welcome, John, Welcome, Mary) Such dynamic behavior causes issues when using the Object Repository, as static properties can no longer be relied on for object recognition.

Descriptive Programming (DP) enables you to handle these objects by:

- Dynamically constructing property-value pairs
- Using regular expressions to match patterns instead of exact values

This removes the dependency on fixed object names in the Object Repository and increases flexibility.

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?

- **Synchronization** in UFT refers to ensuring that test execution is aligned with the speed of the application under test (AUT)**.** In real-time scenarios, the AUT may take time to load pages, render elements, or respond to user actions. If UFT tries to interact with an object before it becomes available**,** the test will fail**,** even though the application is functioning correctly.

3. Synchronization Techniques in UFT

UFT provides multiple ways to handle synchronization:

A. WaitProperty Method (Recommended)

Waits until a specific object property achieves an expected value, or the timeout is reached.

Syntax:

Object.WaitProperty("property name", "expected value", timeout in milliseconds)

Example:

Browser("MyApp").Page("LoginPage").WebButton("Login").
WaitProperty "enabled", True, 10000

B. Exist Method

Checks if the object exists within a given time.

Syntax:

Object.Exist(timeout in seconds)

C. Synchronization Point (GUI Method)

UFT allows inserting synchronization points during recording. **Steps:**

- Go to Insert → Synchronization Point
- Click on the object that should be synchronized
- UFT inserts a WaitProperty line into the script automatically

D. Smart Wait (for Web Applications)

Smart Wait is a feature that enables UFT to automatically wait for a web page's dynamic elements (AJAX, JavaScript rendering) to finish before proceeding. **Enable via:**

- Tools → Options → GUI Testing → Web → Smart Synchronization

E. Hard Coded Wait (Not Recommended)

Pauses execution for a fixed amount of time regardless of object status.

Syntax:

```
Wait 5   'Waits for 5 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.

```
'==============================
' Sync and Wait Example in UFT
'==============================

' Step 1: Launch Browser and Navigate to a page
SystemUtil.Run "iexplore.exe", "https://example.com"

' Step 2: Wait for browser to load completely
Browser("micClass:=Browser").Sync     ' Dynamic wait (waits for page ready)

' Step 3: Optional static wait for UI elements
Wait(3)  ' Wait for 3 seconds (can be adjusted)

' Step 4: Click a button after page is ready
Browser("micClass:=Browser").Page("micClass:=Page").WebButton("html
id:=submitBtn").Click

' Step 5: Report success
Reporter.ReportEvent micPass,
```

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

1. Understanding the Problem

When working with slow-loading applications or pages that load content dynamically (using AJAX, JavaScript, etc.), there is a high chance that UFT may try to interact with objects before they are fully available or ready. This can lead to:

- Object not found errors
- Failed validations or clicks
- Unreliable test results

2. Handling Synchronization Issues Effectively

UFT offers several synchronization strategies that ensure your test waits for the application to reach the desired state before performing any action.

A. Use WaitProperty for Specific Conditions

Wait until a specific object property (like visible, enabled, etc.) matches an expected value.

B. Use Exist to Wait for Object Appearance

Check if the object becomes available within a specified time before taking action.

C. Enable Smart Synchronization (for Web Apps)

Smart Synchronization in UFT automatically waits for web page elements to load and become interactive. To Enable:

- Go to Tools → Options → GUI Testing → Web → Smart Synchronization
- Check: "Enable Smart Synchronization"

D. Use Synchronization Points (Recording Method) During test recording:

- Insert → Synchronization Point
- Click on the object to wait for
- UFT will generate a WaitProperty step automatically

E. Avoid Wait Statements (Use Sparingly)

Hard-coded waits like Wait 5 pause execution for a fixed duration regardless of object readiness. Use this only when necessary, e.g., waiting for a progress bar animation.

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?

  Why Exception Handling is Important in UFT

  During automated test execution, unexpected events like:

- JavaScript pop-ups
- Application alerts
- Security warnings
- Unexpected errors or dialogs

  ...can interrupt the flow of your test and cause it to fail.

  To ensure smooth test execution, you should use exception handling to:

- Detect such interruptions

- Handle them gracefully (e.g., click OK or close the dialog)
- Continue or exit the test in a controlled manner

2. Exception Handling Techniques in UFT

A. Using Recovery Scenarios (GUI-Based Approach)

A **Recovery Scenario** in UFT is a predefined set of instructions that tell UFT how to respond when an unexpected event occurs. **Steps to Create:**

1. Go to Resources → Recovery Scenario Manager
2. Click New → Create a new scenario
3. Define the Trigger Event (e.g., pop-up window, object state, error)
4. Specify the Recovery Operation (e.g., click "OK", close the window) 5. Define Post-Recovery Action (e.g., repeat step, proceed to next)
6. Associate the recovery file with your test under:
   File → Settings → Recovery

B. Using VBScript On Error Resume Next (Script-Based Approach) This approach allows for inline error handling in your test script.

Example: Handling Alert Popup

```
On Error Resume Next
If Browser("MyApp").Dialog("text:=.*Alert.*").Exist(2) Then
   Browser("MyApp").Dialog("text:=.*Alert.*").WinButton("text:=OK").Click End If
On Error GoTo 0
```

C. Using Exist and Conditional Checks

This is a safer, control-based method and works well for UI-based pop-ups or unexpected screens.

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?

1. How UFT Generates Test Results

When you execute a test in UFT (Unified Functional Testing), it automatically generates a detailed test result report, also known as the run results report. This report contains information about:

- Each step executed
- Passed/failed status
- Screen captures
- Error messages
- Custom messages
- Iteration-wise results (for data-driven tests)

UFT saves this report in a structured HTML/XML format in a designated results folder.

2. Where the Results Are Stored

By default, the result files are stored in a folder named: You can customize the location in:

- File → Settings → Run → Results Location

3. How to View the Test Results in UFT Once the test run is complete:

Go to View → Test Results

Or click on the Test Results icon on the toolbar

The Test Results Viewer provides a structured view, including:

- Test Summary
- Step-by-step details
- Time taken per step
- Error messages with descriptions
- Object snapshots (if enabled)

- **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?

The Test Results tab (a.k.a. Results Viewer) appears automatically after a test run (unless disabled). It gives you a complete execution summary, which includes:

- Test status: Passed / Failed / Warning
- Step-by-step execution logs
- Reporter messages
- Errors with descriptions
- Iteration results

Run-Time Data Table

The Run-Time Data Table is a temporary version of the Data Table created and updated during execution. It shows:

- Actual input data used in each iteration
- Any values set or modified by the script using DataTable.Set or DataTable.Value
- A snapshot of the Global and Action-specific data as it was during test execution

How to Use Them to Debug a Failing Test

Step 1: Use the Test Results Tab

- Open Test Results Viewer
- Identify which step failed
- Read the error message and object identification failure
- Check custom logs for logic-related info

Step 2: Open Run-Time Data Table

- Go to the Results Folder → open Default.xls
- Check the actual values that were used during the failed step
- Confirm if incorrect data caused the failure (e.g., wrong username/password)

- **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.

```
'=========================
' Custom Report Generator
'=========================

' Set the report file path Dim
reportFilePath
reportFilePath = "C:\Users\Ananya\Desktop\UFT_Custom_Report.txt"

' Create or open the report file for writing
Set fso = CreateObject("Scripting.FileSystemObject")
Set reportFile = fso.CreateTextFile(reportFilePath, True)

' Write header
reportFile.WriteLine "----------------------------" reportFile.WriteLine "
UFT Custom Test Report" reportFile.WriteLine "----------------------------
" reportFile.WriteLine "Test Execution Time: " & Now
reportFile.WriteLine ""

'======================
```

```
' STEP 1 - Launch Notepad
'======================
On Error Resume Next
SystemUtil.Run "notepad.exe" If
Err.Number = 0 Then
   reportFile.WriteLine "Step 1: Launch Notepad - PASS" Else
   reportFile.WriteLine "Step 1: Launch Notepad - FAIL (" & Err.Description & ")" End If
Err.Clear

'======================
' STEP 2 - Type Text
'======================
Wait(2)
Window("Notepad").WinEditor("Edit").Set "This is a sample automation report demo."
If Window("Notepad").WinEditor("Edit").GetROProperty("text") = "This is a sample
automation report demo." Then
   reportFile.WriteLine "Step 2: Type Text - PASS"
```