***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 an automated functional testing tool developed by Micro Focus. It is designed for functional, regression, and GUI (Graphical User Interface) testing of applications. UFT provides a comprehensive platform for automating the testing of both web-based and desktop applications.

UFT combines the functionality of QTP (QuickTest Professional) with newer features, offering an integrated testing environment with support for testing across various technologies, including web, desktop, mobile, and enterprise applications.

**Difference between UFT, Selenium, and QTP**:

1. **UFT vs Selenium**:
   * **UFT**: A commercial, integrated solution that provides advanced features like object recognition, keyword-driven testing, and support for a wide range of applications (desktop, web, mobile). UFT has a more user-friendly, record-and-playback interface, and it includes built-in reporting and detailed debugging tools.
   * **Selenium**: An open-source, web-based automation tool. Selenium focuses exclusively on web applications and provides greater flexibility for developers through its scripting language support (Java, Python, C#, etc.). Selenium lacks out-of-the-box capabilities for non-web applications (e.g., desktop, mobile apps), which UFT supports.
2. **UFT vs QTP**:
   * **QTP (QuickTest Professional)**: The predecessor to UFT, QTP was designed for functional and regression testing with a focus on web applications. It has been integrated into UFT, which provides a more robust solution with added features, improved support for different application types, and advanced reporting and debugging capabilities.
3. **Key differences**:
   * **Cost**: UFT is a paid, commercial tool, while Selenium is open-source and free.
   * **Technology Coverage**: UFT supports a broader range of technologies (desktop, mobile, web) compared to Selenium, which is limited to web testing.
   * **Ease of Use**: UFT has a more user-friendly interface with record-and-playback features, making it easier for non-developers to use. Selenium, however, requires coding expertise for scripting.

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

**Key Features of UFT**:

1. **Record-and-Playback**: UFT allows users to record actions performed on an application and then play them back for automated testing, simplifying the test creation process, especially for non-technical users.
2. **Keyword-Driven Testing**: UFT enables the creation of test scripts using keywords that represent actions and data, making it easier to develop tests without extensive programming knowledge.
3. **Data-Driven Testing**: UFT allows for testing with multiple sets of input data by linking test scripts with external data sources (e.g., Excel files, databases), enabling robust test case coverage.
4. **Object Recognition**: UFT uses advanced object recognition techniques to identify UI elements in web and desktop applications. It can work with a wide variety of object types, ensuring reliable interaction with the application during tests.
5. **Integrated Debugging**: UFT offers integrated debugging tools to allow testers to step through their test scripts, check variable values, and troubleshoot issues.
6. **Cross-Platform Support**: UFT can automate testing across various platforms such as web, desktop, mobile, and SAP applications.
7. **Extensive Reporting**: UFT generates detailed reports on the test execution, providing insights into test steps, results, and any failures encountered.
8. **Support for Multiple Scripting Languages**: UFT allows the use of VBScript as the default scripting language but also supports other languages, such as JavaScript.
9. **Integration with ALM/QC**: UFT integrates seamlessly with Micro Focus ALM (Application Lifecycle Management) and Quality Center (QC) for test management, version control, and defect tracking.

**How UFT supports functional, regression, and GUI testing**:

1. **Functional Testing**:
   * UFT helps automate functional tests by interacting with the application’s user interface and verifying that the application’s features and functions behave as expected. Testers can create scripts that perform various actions (e.g., clicking buttons, entering data, validating messages) and verify the correct results.
2. **Regression Testing**:
   * UFT is effective for regression testing because it automates the testing of critical application functionalities after code changes or updates. Since UFT supports data-driven testing, testers can quickly execute the same set of tests with different data to ensure that the changes do not break existing features.
3. **GUI Testing**:
   * UFT performs GUI testing by simulating user interactions with the graphical user interface. It can recognize and interact with various GUI elements (buttons, text fields, dropdowns, etc.) and verify that they function as expected. UFT’s object recognition feature ensures accurate identification and interaction with these elements, even when the UI changes slightly.

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

UFT uses an **object recognition model** that identifies and interacts with different types of objects during test execution. The objects that UFT can recognize are categorized based on the technology or framework of the application being tested.

**Types of objects that UFT can recognize**:

1. **Web Objects**:
   * These are UI elements found in web applications, such as buttons, links, text fields, checkboxes, and dropdown lists.
   * **Examples**:
     + Browser: Represents the browser window.
     + Page: Represents a web page.
     + Link: Represents a hyperlink on a webpage.
     + Button: Represents a clickable button on a page.
2. **Desktop Objects**:
   * These include elements found in desktop applications, such as buttons, menus, windows, and controls.
   * **Examples**:
     + Window: Represents a desktop application window.
     + Button: Represents a clickable button on the desktop application.
     + Edit: Represents a text field or input box.
3. **Java Objects**:
   * UFT recognizes objects in Java-based applications, such as buttons, lists, tables, and text fields.
   * **Examples**:
     + Button: A button in a Java application.
     + TextField: A text box where users can input data.
4. **ActiveX Objects**:
   * These are controls used in web and desktop applications, such as embedded ActiveX controls.
   * **Examples**:
     + ActiveX Control: An embedded ActiveX control in a web or desktop application.
5. **SAP Objects**:
   * UFT provides special support for automating SAP applications. It recognizes SAP GUI elements like buttons, fields, and tables.
   * **Examples**:
     + SAPGuiWindow: Represents a SAP window.
     + SAPGuiButton: Represents a clickable button in SAP.
6. **Mobile Objects**:
   * UFT supports mobile application testing and can recognize mobile-specific objects, such as buttons, text fields, and checkboxes, on both Android and iOS platforms.
   * **Examples**:
     + MobileButton: Represents a button on a mobile device.
     + MobileTextBox: Represents a text box on a mobile device.
7. **Web Services/REST/SOAP Objects**:
   * UFT can recognize web services and API requests for automating testing of web services.
   * **Examples**:
     + WebService: Represents a web service that is being tested.
     + SOAPRequest: Represents a SOAP web service request.
8. **Custom Objects**:
   * These are user-defined objects or third-party controls that UFT can recognize by extending its object recognition capabilities.
   * **Example**:
     + Custom controls developed within the application can be added to UFT’s object repository for recognition.

### **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. **Launch UFT**:
   * Open UFT (Unified Functional Testing) on your machine.
2. **Start a New Test**:
   * Click on "File" → "New" → "Test" to create a new test.
   * Name the test, e.g., **NotepadTest**.
3. **Recording the Test**:
   * Click on the "Record" button (a red circle) to start recording the test.
   * UFT will prompt you to select the application you want to test. Choose **Notepad** from the list of available applications.
     + If Notepad is not listed, you can use the "Windows-based" add-in.
4. **Perform Actions in Notepad**:
   * **Open Notepad**: You will have to open Notepad manually if UFT does not launch it automatically.
   * **Type Text**: In the Notepad application, type a simple message, such as "Hello, UFT!".
   * **Save the File**:
     + Press **Ctrl + S** to bring up the save dialog.
     + In the Save dialog, enter a file name like **TestFile.txt**, and click "Save".
5. **Stop Recording**:
   * After completing the actions (typing the text and saving the file), click on the "Stop" button in UFT (a square button) to stop recording.
6. **Review and Edit the Recorded Script**:
   * UFT will automatically generate the script in VBScript format. You can review the code for accuracy and make any necessary changes.

Example script:

vbscript

Copy code

' Launch Notepad

Set NotepadApp = CreateObject("WScript.Shell")

NotepadApp.Run "notepad.exe"

Wait(2) ' Wait for Notepad to open

' Type text into Notepad

NotepadApp.SendKeys "Hello, UFT!"

' Save the file

NotepadApp.SendKeys "^s" ' Ctrl + S

Wait(1)

NotepadApp.SendKeys "TestFile.txt"

NotepadApp.SendKeys "{ENTER}"

1. **Run the Test**:
   * After reviewing and making sure the script is correct, click on the "Run" button (green triangle) to execute the test.
2. **Verify Results**:
   * Ensure that Notepad opens, the text is typed correctly, and the file is saved in the correct location (typically in the default directory for saving files).

**Explanation:**

* **CreateObject("WScript.Shell")**: This creates an instance of Windows Script Host Shell to interact with Windows applications (like Notepad).
* **SendKeys**: This simulates keystrokes (e.g., typing text and pressing keys like Ctrl + S).

**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.**

**UFT Script for Google Search:**

1. **Launch UFT**:
   * Open UFT (Unified Functional Testing) on your machine.
2. **Start a New Test**:
   * Click on "File" → "New" → "Test" to create a new test.
   * Name the test, e.g., **GoogleSearchTest**.
3. **Recording the Test** (Optional but helpful):
   * Click on the "Record" button (red circle) to start recording.
   * Choose the **Web** add-in and select a web browser, such as Google Chrome or Internet Explorer.
4. **Perform Actions**:
   * Open the web browser and go to the URL: www.google.com.
   * In the Google search box, type a search term (e.g., "UFT tutorial").
   * Press **Enter** to initiate the search.
5. **Stop Recording**:
   * After completing the actions (opening the browser, entering the search term, and pressing Enter), click on the "Stop" button in UFT.
6. **Review and Edit the Script**:
   * UFT will generate the test script automatically in VBScript format. Below is a simple script to perform the described actions.

Example script:

' Launch a browser and navigate to Google

Browser("title:=.\*Google.\*").Navigate "http://www.google.com"

' Wait for the page to load

Browser("title:=.\*Google.\*").Page("title:=.\*Google.\*").Sync

' Find the Google search box and enter a search term

Browser("title:=.\*Google.\*").Page("title:=.\*Google.\*").WebEdit("name:=q").Set "UFT tutorial"

' Press Enter to perform the search

Browser("title:=.\*Google.\*").Page("title:=.\*Google.\*").WebEdit("name:=q").Press

1. **Run the Test**:
   * After reviewing and ensuring the script is correct, click the "Run" button (green triangle) to execute the test.
2. **Verify Results**:
   * Ensure that the browser opens, navigates to Google, the search term is entered, and the search results are displayed correctly.

**Explanation of Key Methods:**

* **Browser("title:=.\*Google.\*")**: This identifies the browser window with the title that contains "Google".
* **Page("title:=.\*Google.\*")**: This identifies the page within the browser with the title that contains "Google".
* **WebEdit("name:=q")**: This identifies the search input field on Google’s homepage (the name attribute for the search box is "q").
* **Set**: This method is used to set the value of the WebEdit (text box).
* **Press**: This simulates pressing the Enter key after typing the search query.

### **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."**

**Object Repository in UFT:**

An **Object Repository** in UFT (Unified Functional Testing) is a storage location for all the objects that the test script interacts with during test execution. These objects can include elements like buttons, text fields, dropdowns, links, etc., within the application under test. The repository holds the **properties** and **methods** for each object, which UFT uses to identify and interact with the objects during testing.

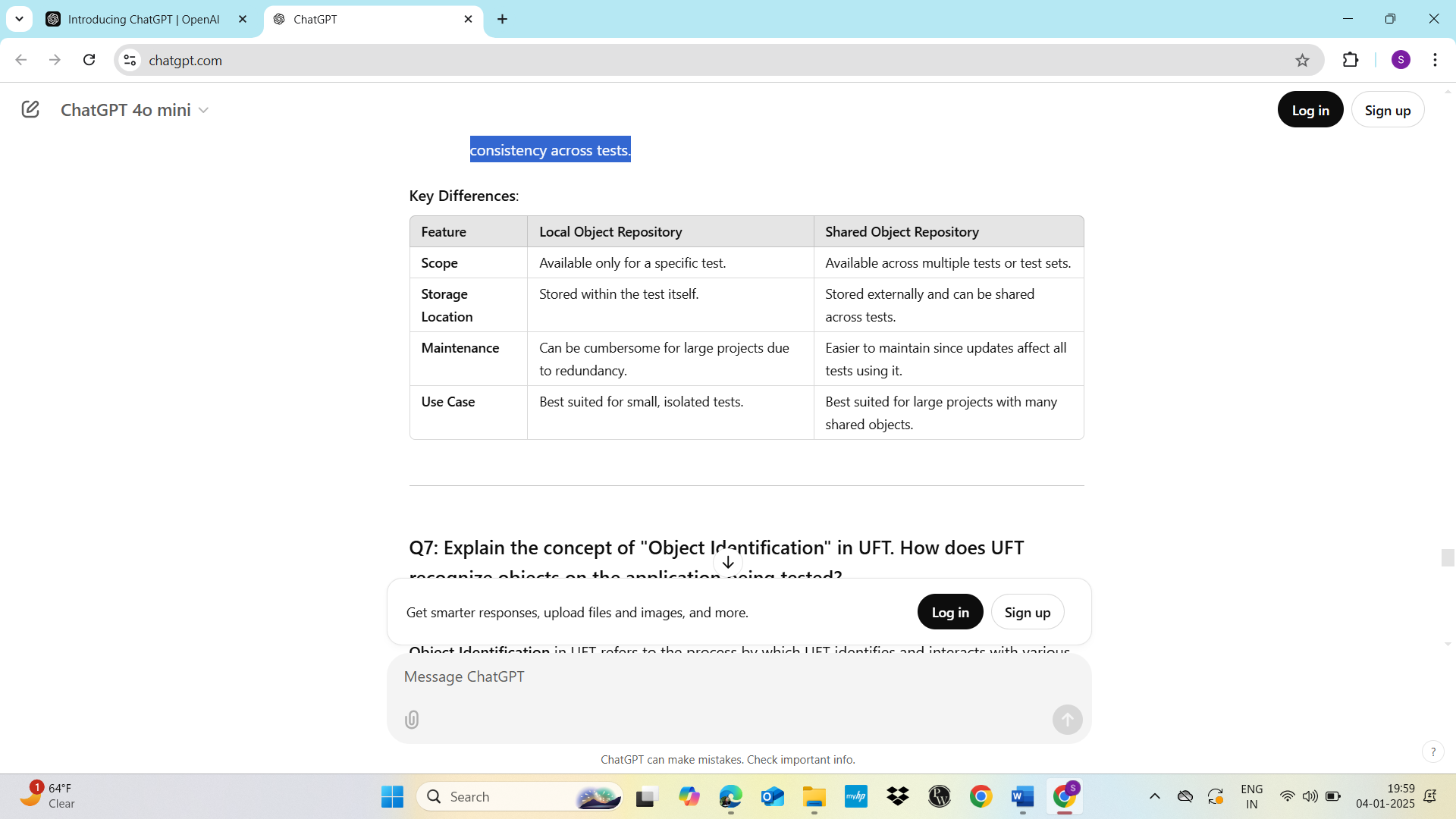
There are two types of Object Repositories in UFT: **Local Object Repository** and **Shared Object Repository**.

**Local Object Repository:**

* **Definition**: A Local Object Repository is an object repository that is specific to a single test. The objects in the Local Object Repository are available only for that particular test.
* **Usage**: It is useful when you are testing a small application or working on a single test where the objects are not used by other tests.
* **Characteristics**:
  + The objects are stored within the test itself (test folder).
  + The test script references the local repository when interacting with the application.
  + Changes made to the objects in the local repository affect only the specific test in which they are stored.

**Shared Object Repository:**

* **Definition**: A Shared Object Repository is an object repository that can be used by multiple tests. It allows for the reuse of objects across different tests or test sets.
* **Usage**: It is useful in larger projects where multiple tests use the same objects, ensuring consistency and reducing the need for redundant object creation.
* **Characteristics**:
  + The objects are stored externally, outside of the test (in a central location).
  + Multiple tests can reference and use the same shared repository.
  + Changes made to the objects in the shared repository affect all the tests that use it.
  + The shared repository can be updated or modified centrally, making it easier to maintain consistency across tests.



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

**Object Identification in UFT:**

**Object Identification** in UFT refers to the process by which UFT identifies and interacts with various objects within an application during test execution. It involves recognizing objects like buttons, text fields, checkboxes, links, etc., within the user interface (UI) of the application being tested.

UFT uses **properties** of these objects to identify them uniquely and interact with them. The **object repository** holds the description of each object, which includes a set of properties (such as name, class, id, text, etc.) that are used to identify the object during testing. These properties are compared against the actual properties of the UI element in the application under test.

**How UFT Recognizes Objects:**

1. **Object Repository**:
   * UFT uses the **Object Repository** to store the properties and characteristics of objects that are to be interacted with. The repository contains a list of objects with their corresponding properties that uniquely identify them.
2. **Object Properties**:
   * Every object in the application has certain **properties** associated with it (e.g., name, ID, class, text, etc.). UFT uses these properties to uniquely identify and interact with objects.
   * UFT checks the actual properties of an object at runtime and compares them with the properties stored in the Object Repository to recognize and act on it.
3. **Smart Identification**:
   * **Smart Identification** is a fallback mechanism that UFT uses when the standard object identification fails (e.g., if an object’s properties change dynamically during the execution). Smart Identification identifies objects using a combination of mandatory and optional properties.
   * **Mandatory Properties**: These are essential for identifying the object (e.g., name, id).
   * **Optional Properties**: These help refine the identification process but are not required to identify the object (e.g., text, type).
   * If the standard identification mechanism fails, Smart Identification uses these properties to locate the object.
4. **Object Identification Process**:
   * **Object Spy**: UFT provides an "Object Spy" tool that allows testers to inspect an object’s properties. Testers can use Object Spy to identify the properties of any object in the application under test.
   * **Checkpoints**: UFT can insert checkpoints into the test to verify that the object is present and has the expected properties during test execution.
5. **Descriptive Programming**:
   * In cases where the object is not present in the Object Repository, **Descriptive Programming** can be used to directly specify the properties of the object in the test script. This allows UFT to identify the object dynamically during runtime without relying on the repository.
6. **Regular Expressions**:
   * UFT supports the use of **regular expressions** to handle dynamic or changing object properties. This helps when the object properties change (e.g., a dynamic ID or changing text), allowing UFT to recognize objects based on a pattern rather than a fixed value.

**Example of Object Identification:**

Consider a scenario where we have a button with the following properties:

* class: Button
* name: Submit
* text: Submit

UFT would check these properties to identify the button during the test. If any of the properties change dynamically, Smart Identification would try to identify the button using other properties or a combination of mandatory and optional properties.

**Example code using Descriptive Programming**:

' Descriptive Programming to identify a Submit button

Set submitButton = Browser("title:=.\*").Page("title:=.\*").ChildObjects("micclass:=Button", "name:=Submit")

submitButton(0).Click

In this example, UFT uses Descriptive Programming to identify the Submit button by matching its name and micclass property. This eliminates the need for storing the object in the Object Repository.

**Summary of Object Identification Process in UFT:**

* **Object Repository**: Holds the properties of objects.
* **Object Properties**: UFT uses a set of properties (name, class, ID, etc.) to identify objects.
* **Smart Identification**: Used when standard identification fails, utilizing mandatory and optional properties.
* **Descriptive Programming**: Allows identification of objects using properties directly in the script.
* **Regular Expressions**: Enables dynamic identification of objects with changing properties.

### **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:**

A **checkpoint** in UFT (Unified Functional Testing) is a verification point that helps you compare the expected behavior of an application with its actual behavior during test execution. It captures the current state of an object or application and compares it with the expected value to determine if the test passes or fails. Checkpoints allow you to validate that the application is functioning as expected.

There are several types of checkpoints in UFT:

* **Text Checkpoint**: Verifies that a specified text appears in a web page or an application.
* **Image Checkpoint**: Verifies the appearance of an image on a page.
* **Database Checkpoint**: Verifies the data in a database.
* **Table Checkpoint**: Verifies the data in a table.
* **Page Checkpoint**: Verifies the properties of a web page.
* **Accessibility Checkpoint**: Verifies accessibility standards.

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

1. **Record or Open the Test**:
   * Open UFT and either start recording or open an existing test for the web application you want to test.
2. **Insert a Text Checkpoint**:
   * Once the test is recorded or the script is open, identify the web page or object that contains the text you want to verify.
   * Right-click on the object (e.g., a web element that contains text) in the **Keyword View** or **Expert View**.
   * In the Keyword View, select **Insert** → **Checkpoint** → **Text Checkpoint**.
3. **Configure the Checkpoint**:
   * In the **Text Checkpoint** dialog box, specify the text you want to check on the web page.
   * Set the expected text value (e.g., "Welcome to UFT Testing").
4. **Run the Test**:
   * After adding the checkpoint, run the test. UFT will compare the actual text with the expected value and report whether the test passes or fails based on the comparison.

**Sample Script to Add a Text Checkpoint:**

' Navigate to a web page

Browser("title:=.\*Google.\*").Navigate "http://www.google.com"

' Wait for the page to load

Browser("title:=.\*Google.\*").Page("title:=.\*Google.\*").Sync

' Insert a Text Checkpoint to verify the presence of specific text

Browser("title:=.\*Google.\*").Page("title:=.\*Google.\*").Check Checkpoint("TextCheckpoint")

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

**Standard Checkpoints:**

* **Definition**: Standard Checkpoints are used to verify properties of objects in the application, such as text, images, tables, and pages.
* **Usage**: They can be applied to validate the UI elements, ensuring that the text, appearance, and functionality of objects are correct.
* **Example Types**:
  + **Text Checkpoints**: Verifies that a specific text appears on a page.
  + **Image Checkpoints**: Verifies that an image is displayed correctly.
  + **Table Checkpoints**: Verifies the data in a table on a page.
  + **Page Checkpoints**: Verifies the properties of a web page, such as its title or URL.

**When to Use Standard Checkpoints**:

* You would use Standard Checkpoints when testing the **UI elements** of an application. For example, when verifying if the text displayed on a button matches the expected value or if an image is displayed correctly.

**Example Use Case**:

* Verifying if a "Login Successful" message is displayed after a user logs in successfully.

**Database Checkpoints:**

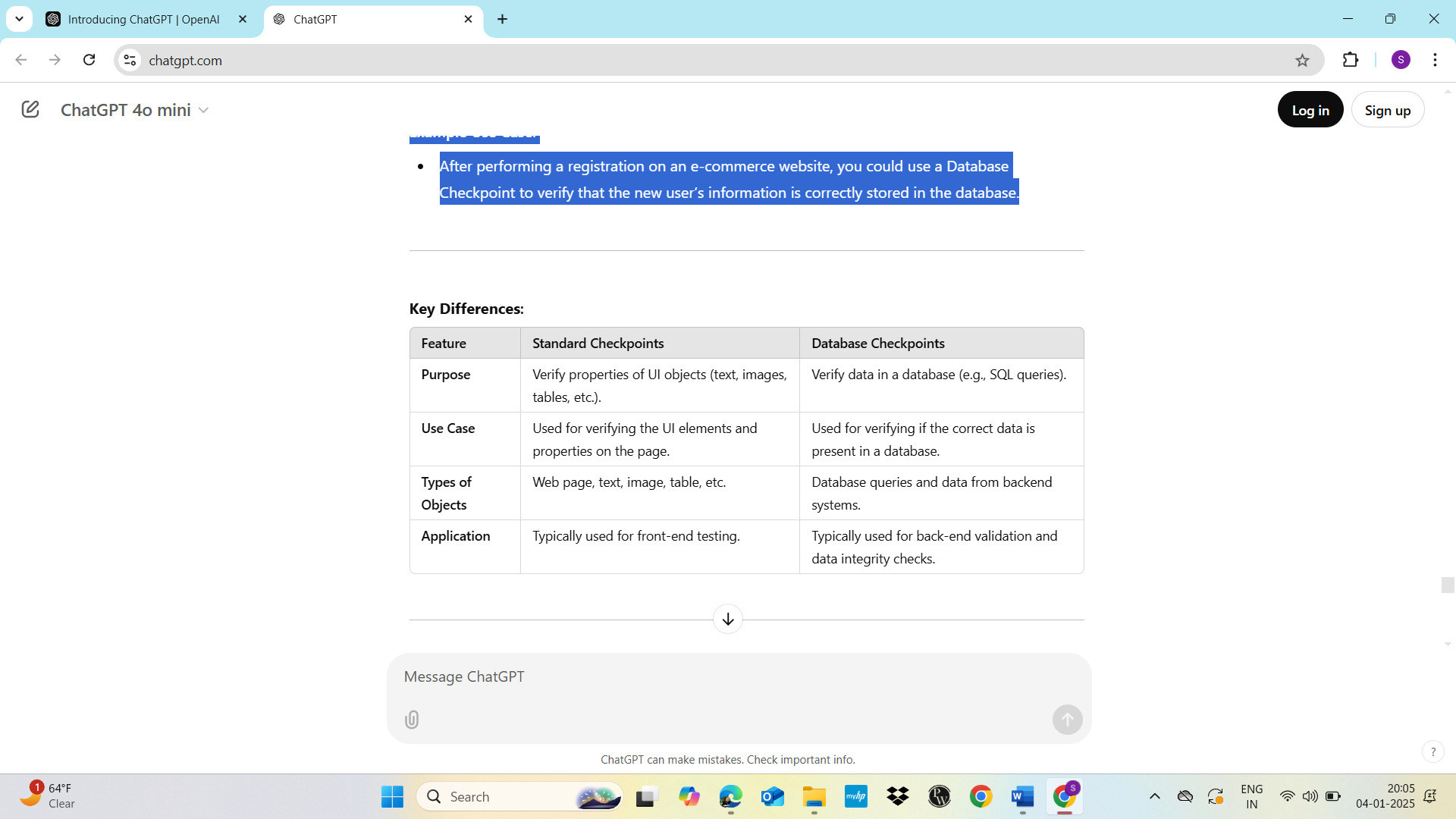
* **Definition**: A **Database Checkpoint** is used to verify the data in a database (such as an Oracle, SQL, or MySQL database). It allows you to query the database during the test to validate that the data is correct.
* **Usage**: Database Checkpoints are used when you need to ensure that the data retrieved from the database (e.g., during an application’s execution) matches expected values.
* **Example**: Verifying that a new user’s record has been successfully added to a database after registering on the application.

**When to Use Database Checkpoints**:

* Use a Database Checkpoint when you need to validate data from a database as part of your test case. For instance, you may want to verify if the data displayed on the UI matches the data in the backend database.

**Example Use Case**:

* After performing a registration on an e-commerce website, you could use a Database Checkpoint to verify that the new user’s information is correctly stored 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.**

**Handling Dynamic Objects in UFT:**

Dynamic objects are objects that change their properties (such as text, name, ID, or position) during the execution of a test. UFT provides several methods for handling dynamic objects effectively.

**Techniques to Handle Dynamic Objects**:

1. **Using Regular Expressions**: UFT supports **Regular Expressions (RegEx)** to handle dynamic properties that may change over time (e.g., changing text or dynamic IDs). By using regular expressions, you can match patterns rather than exact values.
2. **Descriptive Programming**: **Descriptive Programming** allows you to dynamically locate objects during test execution without using the Object Repository. You can use it to handle objects with dynamic properties by specifying properties dynamically in your test scripts.
3. **Smart Identification**: If an object’s identification fails, UFT’s **Smart Identification** mechanism helps identify the object by looking for a combination of mandatory and optional properties.
4. **Using "ChildObjects"**: You can use the **ChildObjects** method when handling dynamic child objects (e.g., buttons that change text), where you search for objects based on partial matching of dynamic attributes.

**Example of Handling Dynamic Buttons:**

Imagine a scenario where a button changes its text based on user interactions (e.g., "Submit" changes to "Processing").

1. **Using Regular Expressions**: If the button’s text changes dynamically, you can use a regular expression to match the changing text.

**Script Example**:

' Identify a button with dynamic text using RegEx

Set dynamicButton = Browser("title:=.\*").Page("title:=.\*").WebButton("text:=.\*Submit.\*")

' Click the button

dynamicButton.Click

In this example:

* + The regular expression .\*Submit.\* is used to match any button whose text contains the word "Submit" (even if it changes to "Processing").

1. **Using Descriptive Programming**: If the button's properties change dynamically, you can use Descriptive Programming to search for the button based on partial or changing attributes.

**Script Example**:

' Descriptive Programming to identify a button with dynamic text

Set dynamicButton = Browser("title:=.\*").Page("title:=.\*").ChildObjects("micclass:=WebButton", "text:=Submit")

dynamicButton(0).Click

In this example, Descriptive Programming is used to identify the dynamic button by the micclass (button) and text property, which can change during execution.

**Summary:**

* **Dynamic Objects**: Objects whose properties change during runtime, such as text or IDs.
* **Techniques for Handling Dynamic Objects**:
  + **Regular Expressions**: Handle objects with changing properties (e.g., button text).
  + **Descriptive Programming**: Locate dynamic objects by specifying properties directly in the script.
  + **Smart Identification**: UFT's fallback method for dynamic object identification.
  + **ChildObjects**: Useful for handling collections of dynamic objects.

### **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:**

**Parameterization** in UFT refers to the process of substituting hard-coded values in your test scripts with variables that can take different values during execution. This allows you to run the same test with different sets of data, making the test more versatile and reusable.

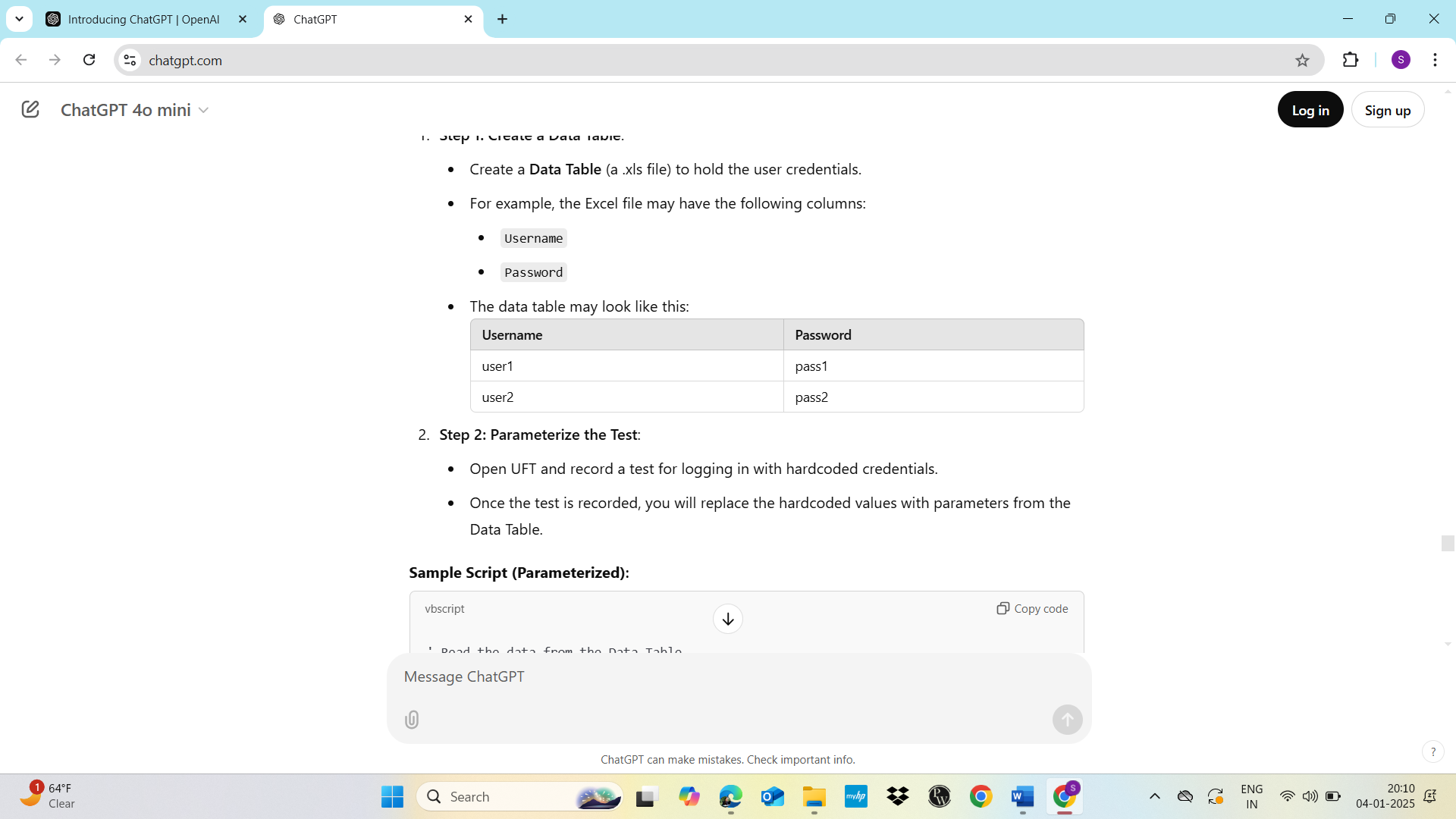
**Importance of Parameterization:**

* **Increases Test Coverage**: With parameterization, you can run the same test with multiple sets of data to cover different scenarios (e.g., valid/invalid credentials, different browsers, etc.).
* **Reduces Script Maintenance**: Instead of modifying the script to test with different data, parameterization allows you to test with different inputs by simply changing the data source.
* **Improves Test Efficiency**: By using data-driven testing, you can perform tests with a variety of input values without manually changing the test case each time.

**How to Parameterize a Test (Using User Credentials):**

In this example, we will demonstrate how to parameterize a test where you input user credentials (username and password) for a login page.

1. **Step 1: Create a Data Table**:
   * Create a **Data Table** (a .xls file) to hold the user credentials.
   * For example, the Excel file may have the following columns:
     + Username
     + Password
   * The data table may look like this:



1. **Step 2: Parameterize the Test**:
   * Open UFT and record a test for logging in with hardcoded credentials.
   * Once the test is recorded, you will replace the hardcoded values with parameters from the Data Table.

**Sample Script (Parameterized):**

vbscript

Copy code

' Read the data from the Data Table

username = DataTable("Username", dtGlobalSheet)

password = DataTable("Password", dtGlobalSheet)

' Navigate to the login page

Browser("title:=LoginPage").Navigate "http://www.example.com/login"

' Wait for the page to load

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

' Enter the username and password from the DataTable

Browser("title:=LoginPage").Page("title:=LoginPage").WebEdit("name:=username").Set username

Browser("title:=LoginPage").Page("title:=LoginPage").WebEdit("name:=password").Set password

' Click the Login button

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

' Add verification points or checks as needed

In this example:

* The script reads values from the **Data Table** (which can be an Excel sheet).
* It uses the DataTable function to retrieve the Username and Password values for each iteration and logs in with the respective credentials.
* The test is now parameterized and can be run with different sets of data without modifying the script.

**Q14: Create a Test That Accepts Input Parameters (e.g., Username and Password) from an Excel File and Performs a Login Using That Data.**

To accept input parameters (such as username and password) from an Excel file and perform a login using that data, you would follow these steps:

1. **Create an Excel File**:
   * Save an Excel file with columns Username and Password.
   * Example Excel file:

| **Username** | **Password** |
| --- | --- |
| testuser | testpass |
| admin | admin123 |

1. **UFT Test Script**: The following UFT script demonstrates how to read the parameters (username and password) from the Excel file and use them for login:

vbscript

Copy code

' Initialize the Data Table (assuming data is stored in the Excel file)

DataTable.Import "C:\path\to\your\datafile.xlsx" ' Provide the correct file path

' Loop through the data in the Data Table

Do While Not DataTable.EOF

' Get the username and password for this iteration

username = DataTable("Username", dtGlobalSheet)

password = DataTable("Password", dtGlobalSheet)

' Navigate to the login page

Browser("title:=LoginPage").Navigate "http://www.example.com/login"

' Wait for the page to load

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

' Enter the username and password from the DataTable

Browser("title:=LoginPage").Page("title:=LoginPage").WebEdit("name:=username").Set username

Browser("title:=LoginPage").Page("title:=LoginPage").WebEdit("name:=password").Set password

' Click the Login button

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

' Add verification points or checks as needed

' Move to the next row of data

DataTable.NextRow

Loop

In this example:

* **DataTable.Import** imports data from the Excel file.
* **DataTable("Username", dtGlobalSheet)** reads the Username value from the current row.
* The test performs a login using each set of credentials from the Excel file, iterating over the data until all rows have been processed.

**Q15: What Are the Different Types of Parameters Available in UFT (e.g., Test, Action, and Data Table Parameters)? Explain Their Use with Examples.**

UFT supports three main types of parameters:

1. **Test Parameters**
2. **Action Parameters**
3. **Data Table Parameters**

**1. Test Parameters:**

* **Definition**: Test parameters are used to pass data to the entire test. These parameters are defined at the test level and can be used throughout the test.
* **Usage**: Test parameters are useful when you want to pass data or variables that affect the entire test execution.
* **Example**:
  + A test parameter could be used to pass the environment URL (like https://staging.example.com or https://prod.example.com) to the test.

**Defining Test Parameters**:

* + Open the test in **Test Flow** view.
  + Right-click on the test and select **Add Parameter**.
  + Provide a name and value for the parameter.

**Sample Script**:

' Retrieve test parameter

envURL = Parameters("EnvURL")

' Navigate to the appropriate environment

Browser("title:=.\*").Navigate envURL

**2. Action Parameters:**

* **Definition**: Action parameters are defined within specific actions and allow you to pass data to that action. These parameters are only accessible within the scope of the action in which they are defined.
* **Usage**: Action parameters are used when you need to reuse actions with different inputs. They are useful in modular testing.
* **Example**:
  + A login action might take a username and password as action parameters to allow different users to log in.

**Defining Action Parameters**:

* + In the **Action** properties, define parameters for the action.

**Sample Script**:

' Define action parameters in the Action's properties

username = Parameters("Username")

password = Parameters("Password")

' Perform login using parameters

Browser("title:=LoginPage").Page("title:=LoginPage").WebEdit("name:=username").Set username

Browser("title:=LoginPage").Page("title:=LoginPage").WebEdit("name:=password").Set password

**3. Data Table Parameters:**

* **Definition**: Data Table parameters are used to pass data stored in a data table (such as an Excel file) to the test or action. Data tables are often used for data-driven testing.
* **Usage**: Data table parameters are used when you need to run tests with multiple sets of data, typically for parameterized tests.
* **Example**:
  + Using **Data Table Parameters** to fetch login credentials stored in a data table.

**Sample Script**:

' Retrieve data from the Data Table

username = DataTable("Username", dtGlobalSheet)

password = DataTable("Password", dtGlobalSheet)

' Use the retrieved data to perform the login action

Browser("title:=LoginPage").Page("title:=LoginPage").WebEdit("name:=username").Set username

Browser("title:=LoginPage").Page("title:=LoginPage").WebEdit("name:=password").Set password

### **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.**

**Action in UFT:** An *action* in UFT (Unified Functional Testing) is a logical component of your test script. It can be considered as a self-contained part of the test that performs a specific task or a group of tasks. Actions allow you to break down your script into manageable, reusable, and organized parts.

There are two types of actions in UFT:

1. **Reusable Action**: This can be called from multiple tests. It allows you to reuse the same functionality without writing the same code multiple times.
2. **Non-Reusable Action**: This is used only within the test in which it is defined and cannot be reused in other tests.

**How Actions Help in Organizing Test Scripts:**

* Actions break down complex tests into simpler, reusable modules.
* You can execute, modify, and debug actions independently of other parts of the test.
* Reusable actions can be shared across tests to maintain consistency and reduce code duplication.

**Example of a Reusable Action for Logging into a Web Application:**

1. **Create a New Action**:
   * In UFT, go to the "Action" menu and click "New Action".
   * Define the name of the action as LoginAction and mark it as reusable.
2. **Code for Reusable Login Action:**

' LoginAction

' This action logs into a web application with the given username and password

' Parameters for login credentials

' Input parameters: username, password

Browser("LoginPage").Page("LoginPage").WebEdit("username").Set "testuser"

Browser("LoginPage").Page("LoginPage").WebEdit("password").Set "testpassword"

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

End Action

1. **Call the Reusable Action in Your Test Script:**

' Test Script

' Main test script that calls the reusable LoginAction

' Launch the browser and open the web application

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

' Call the reusable LoginAction

LoginAction "testuser", "testpassword"

This allows you to log in using the same action across multiple tests.

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

**Function Libraries in UFT:** A *function library* is a collection of functions written in VBScript that can be reused across multiple tests in UFT. These functions typically perform specific tasks or operations, such as interacting with web elements, performing calculations, or executing common test setup actions. Function libraries help in maintaining reusable code and reducing redundancy.

**How to Create and Associate a Function Library with Your Test:**

1. **Create a Function Library:**
   * In UFT, go to "File" → "New" → "Function Library".
   * Write the function code that you want to reuse. For example, a simple function for logging in:

' Function Library Example

Function Login(username, password)

' Function to perform login action

Browser("LoginPage").Page("LoginPage").WebEdit("username").Set username

Browser("LoginPage").Page("LoginPage").WebEdit("password").Set password

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

End Function

1. **Associate the Function Library with a Test:**
   * Open your test in UFT.
   * Go to the "Test" menu → "Settings" → "Resource" tab.
   * Under "Function Libraries," click "Add" and select the function library file you created.
   * Now, your test can use any functions defined in that library.
2. **Call Functions from the Function Library in Your Test:**

' Test Script

' Example of using the Login function from the function library

Login "testuser", "testpassword"

This setup allows you to use functions from the function library without rewriting the code in every test.

**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.**

**Step 1: Create the Function Library**

1. Open UFT and create a new function library (File → New → Function Library).
2. Write the following function in the function library:

' Function Library Example for Adding Two Numbers

Function AddNumbers(num1, num2)

' This function returns the sum of two numbers

AddNumbers = num1 + num2

End Function

**Step 2: Associate the Function Library with Your Test**

1. Open your test in UFT.
2. Go to the "Test" menu → "Settings" → "Resource" tab.
3. Under "Function Libraries," click "Add" and select the function library you just created.

**Step 3: Call the Function in Your Test Script**

' Test Script

' Example of using the AddNumbers function from the function library

Dim result

result = AddNumbers(10, 20) ' This will return 30

' Output the result to the results window

Reporter.ReportEvent micPass, "Addition Result", "The sum of 10 and 20 is " & result

This will display the result of adding 10 and 20 as 30 in the UFT results window.

### **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 (DP) is a way to interact with objects in UFT without relying on the Object Repository. Instead of storing the properties of objects in the Object Repository, you can directly specify the properties of the objects in your script using a descriptive description object. DP is useful when:

* You don’t want to store objects in the Object Repository (for example, when dealing with dynamic objects).
* The object is not available in the Object Repository.
* You need to handle objects dynamically, such as objects with changing attributes.

**When to Use Descriptive Programming:**

* When the object is dynamically generated and its properties change at runtime.
* When you don’t want to use the Object Repository for a specific object.
* For quick and lightweight scripts where you don’t need to maintain an Object Repository.

**Example of Descriptive Programming to Click a "Submit" Button on a Webpage:**

' Descriptive Programming to Click a Submit Button

' Create a description object for the Submit button

Dim objSubmitButton

Set objSubmitButton = Description.Create

objSubmitButton("html tag").Value = "INPUT"

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

objSubmitButton("name").Value = "submitBtn" ' Assuming the button has name "submitBtn"

' Use the description object to click the Submit button

Browser("example.com").Page("HomePage").ChildObjects(objSubmitButton)(0).Click

**Explanation:**

* The Description.Create method creates a description object to store the properties of the Submit button.
* The ChildObjects method retrieves all objects matching the description, and (0) selects the first match.
* The Click method is used to click the button.

**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.).**

**Syntax for Descriptive Programming in UFT:** The basic syntax for Descriptive Programming consists of the following steps:

1. **Create a Description Object:** This object stores the properties of the web element.
2. **Set Properties:** The properties of the object (such as name, type, innertext, id, etc.) are defined using the description object.
3. **Use the Description Object to Interact with the Object:** Use methods like ChildObjects or Object to identify the object and interact with it.

**Descriptive Programming Syntax Example:**

' Create a description object for a link element

Dim objLink

Set objLink = Description.Create

objLink("tagname").Value = "A" ' "A" represents a hyperlink (anchor tag)

objLink("innertext").Value = "Click here" ' The link text

' Use the description object to click the link

Browser("example.com").Page("HomePage").ChildObjects(objLink)(0).Click

**Explanation:**

* Description.Create initializes a description object.
* The properties tagname and innertext are assigned values to uniquely identify the link.
* The ChildObjects method is used to find and interact with the element (i.e., clicking the link).

**Common Properties for Descriptive Programming:**

* tagname: The HTML tag name (e.g., A, DIV, INPUT).
* id: The unique identifier of the element.
* name: The name attribute of the element.
* class: The CSS class attribute.
* innertext: The visible text inside the element.
* type: The type attribute (e.g., button, submit, text).

**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:** Dynamic objects are those whose properties (e.g., ID, name, class, etc.) change during each run or session. UFT handles dynamic objects using Descriptive Programming by identifying the object through multiple properties that might stay constant across executions or by using wildcards.

**Example of Handling Dynamic Link or Button Using Descriptive Programming:**

Assume that a button’s ID changes dynamically based on session, but the name attribute or class remains constant. We can use Descriptive Programming with wildcards or other properties to handle this.

' Create a description object for a dynamic button

Dim objDynamicButton

Set objDynamicButton = Description.Create

objDynamicButton("tagname").Value = "INPUT" ' Button tag

objDynamicButton("type").Value = "submit" ' Button type

objDynamicButton("name").Value = "submitBtn\_\*" ' Dynamic part of name

' Use the description object to click the button

Browser("example.com").Page("HomePage").ChildObjects(objDynamicButton)(0).Click

**Explanation:**

* The name property of the button is assumed to be dynamic, with a wildcard \* at the end ("submitBtn\_\*"). This allows the script to match any button where the name starts with submitBtn\_.
* The ChildObjects method retrieves all objects that match the description, and (0) selects the first match.

**Dynamic Object Handling Strategies in Descriptive Programming:**

1. **Wildcards (\*, ?):**
   * \*: Matches any sequence of characters (e.g., submitBtn\_\*).
   * ?: Matches a single character (e.g., submitBtn\_?).
2. **Using Multiple Properties:**
   * Sometimes a single property like name or id might not be enough. By combining multiple properties, like class, tagname, and type, UFT can identify the object even if one property changes dynamically.
3. **Regular Expressions:**
   * UFT also supports regular expressions in descriptive programming for more flexible matching of dynamic objects.

### **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?**

**Importance of Synchronization in UFT:** Synchronization ensures that UFT waits for the appropriate amount of time before performing actions on objects or elements. This is crucial because:

* Web pages or applications often load slowly due to network issues, backend processing, or dynamic content.
* If UFT performs actions too quickly before an element is fully loaded or available, it might cause the script to fail, leading to errors or incorrect results.
* Synchronization helps maintain the stability of your test script, ensuring that all elements are ready for interaction.

**Different Synchronization Techniques in UFT:**

1. **Implicit Waits:**
   * UFT has a built-in timeout mechanism for waiting for objects to appear in the Object Repository.
   * You can set a global timeout that forces UFT to wait for an object to become available.
2. **Sync Method:**
   * The Sync method waits for a specified period to allow the application to synchronize with UFT before executing the next action.
   * It's particularly useful when you want to wait for the page or object to load before continuing.
3. **Wait Method:**
   * The Wait method pauses the execution of the script for a specified amount of time (in seconds).
   * It's a simple and effective way to wait for an element to appear, especially for dynamic content.
4. **Exist Method:**
   * The Exist method checks whether an object exists or not. UFT will wait until the object is available or the timeout is reached.
   * It is useful when the script needs to wait for an element to exist on the page.
5. **WaitProperty Method:**
   * The WaitProperty method is used to wait until a specific property of an object reaches a defined value.
   * It’s useful for waiting for dynamic content, such as a changing button label or status.
6. **Custom Synchronization Using Loops:**
   * You can write custom code using a loop to check if an object is ready or available, and then proceed with the next action when the condition is met.

**Examples of Synchronization Techniques in UFT:**

* Sync: Browser("Browser").Sync
* Wait: Browser("Browser").Page("Page").Wait(5) (Wait for 5 seconds)
* Exist: Browser("Browser").Page("Page").WebButton("Submit").Exist(10) (Wait for up to 10 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.**

Here is an example script that uses both the Sync method and the Wait method to ensure UFT waits for a page to load before clicking a button:

' Launch the browser and navigate to the page

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

' Sync method to ensure the page is fully loaded before proceeding

Browser("example.com").Sync

' Wait method to wait for the page to load for 10 seconds

Browser("example.com").Page("HomePage").Wait(10)

' Now, click the 'Submit' button after the page is loaded

Browser("example.com").Page("HomePage").WebButton("Submit").Click

' You can add more actions here

**Explanation:**

* The Sync method is used to synchronize the browser object to ensure the page is fully loaded before moving forward.
* The Wait method is used to make UFT wait for 10 seconds to ensure the page has loaded before interacting with any elements (such as the "Submit" button).
* After the synchronization, the script clicks the "Submit" button on the page.

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

When testing a slow application or a page with dynamic content, it is common to encounter synchronization issues where elements are not ready for interaction when UFT attempts to interact with them. Here are several approaches to handle these issues effectively:

1. **Use Explicit Waits (Exist, WaitProperty, Sync):**
   * **Exist Method:** Use the Exist method to check if an element exists on the page before performing any actions. This ensures that UFT will wait for the object to appear before interacting with it.
   * Example:

' Wait for the element to exist (up to 30 seconds)

If Browser("example.com").Page("HomePage").WebButton("Submit").Exist(30) Then

Browser("example.com").Page("HomePage").WebButton("Submit").Click

Else

Reporter.ReportEvent micFail, "Submit Button", "The Submit button did not appear in time."

End If

* + **WaitProperty Method:** This method is useful for waiting until a specific property of an object reaches a desired value.

Browser("example.com").Page("HomePage").WebButton("Submit").WaitProperty "enabled", True, 30

This waits for the "Submit" button to be enabled before clicking.

1. **Use Sync Method:**
   * The Sync method ensures that UFT waits for a page or object to fully load before moving on with the script. It’s particularly helpful for waiting for dynamic elements like AJAX-based content that load after an initial page load.
   * Example:

Browser("example.com").Sync

1. **Implement Custom Wait Loops:**
   * For pages with dynamic content, a custom wait loop can be used to repeatedly check for the availability of elements.
   * Example:

Dim counter

counter = 0

Do While counter < 10

If Browser("example.com").Page("HomePage").WebButton("Submit").Exist Then

Browser("example.com").Page("HomePage").WebButton("Submit").Click

Exit Do

End If

counter = counter + 1

Wait(1) ' Wait for 1 second before checking again

Loop

If counter = 10 Then

Reporter.ReportEvent micFail, "Submit Button", "The Submit button did not appear in time."

End If

1. **Handle Slow Network/Server Delays:**
   * If an application is slow due to network or server issues, increase the Timeout settings in UFT to wait longer for objects to appear.
   * You can adjust the timeout settings under **Tools** → **Options** → **Run** → **Object Timeout**.
2. **Using Wait Method for Fixed Delays:**
   * If you know that the page takes a specific time to load, use the Wait method for a fixed delay. This is simple but can lead to inefficient tests if not used carefully.
   * Example:

Browser("example.com").Page("HomePage").Wait(15) ' Wait for 15 seconds

Browser("example.com").Page("HomePage").WebButton("Submit").Click

1. **Use Object Synchronization for Dynamic Elements:**
   * If your application has dynamically changing content (e.g., an element that changes its ID), use dynamic descriptive programming with wildcards (\*, ?) to handle these dynamic objects.

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

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

**Exception Handling in UFT:** Exception handling helps manage unexpected pop-ups or alerts during test execution. UFT provides the following ways to handle such exceptions:

**1. Recovery Scenarios:**

* **Recovery Scenarios** in UFT are designed to handle unexpected events (like alerts or pop-ups). You can create a recovery scenario to specify actions (e.g., clicking "OK") when an alert appears.
* **Steps to set up:**
  + Go to **Tools → Recovery Scenarios** to create a new recovery scenario.
  + Define the event (e.g., a pop-up appears).
  + Associate it with the test.

**2. VBScript Error Handling:**

You can use **On Error Resume Next** and **Err** object to handle errors.

**Example:**

On Error Resume Next ' Ignore errors

' Attempt to click the button

Browser("example.com").Page("HomePage").WebButton("Submit").Click

' Check if alert exists and handle it

If Browser("example.com").Dialog("Alert").Exist Then

Browser("example.com").Dialog("Alert").Click "OK"

End If

If Err.Number <> 0 Then

MsgBox "Error Occurred: " & Err.Description

End If

On Error GoTo 0 ' Reset error handling

**3. Using the Exist Method:**

You can use the **Exist** method to check if a pop-up appears before attempting to interact with it.

**Example:**

If Browser("example.com").Dialog("Alert").Exist(5) Then

Browser("example.com").Dialog("Alert").Click "OK"

End If

**Conclusion:**

* **Recovery Scenarios** provide a simple way to handle unexpected alerts.
* **VBScript error handling** (using On Error Resume Next and Err) allows programmatic error management.
* **Exist method** helps detect pop-ups before interacting with them.

### **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:** After running a test, UFT generates a detailed **Test Results** file that contains the outcomes of each step in the test script. The result file includes:

* **Status of each step** (Pass, Fail, Done, or Warning).
* **Error messages** or any **failure details** for failed steps.
* **Snapshots** of the screen when certain checkpoints or failures occur.
* **Log information** for each step, such as timestamps and comments.

These results are stored in an **.html** or **.xml** format and are automatically opened in UFT when the test execution is complete. This makes it easy to review the test execution.

**How to View and Analyze Test Results:**

1. **Test Results Window:**
   * After running a test, UFT opens the **Test Results** window where you can view a summary of the test execution.
   * The results window displays a **tree view** of the test steps with status indicators like green (pass), red (fail), yellow (warning), etc.
   * You can drill down into each step to see detailed log information.
2. **Detailed Log:**
   * For each test step, you can click to view additional logs, including **screenshot attachments** if UFT captured them during execution.
3. **Report Section:**
   * The **Summary** tab shows the overall pass/fail status of the entire test.
   * The **Log** tab provides detailed information for each step, including executed code and any relevant messages.
4. **Using the “Reporter” Object:**
   * UFT allows you to customize the output using the **Reporter** object, providing more control over how test results are logged and reported.

**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 vs Run-Time Data Table:**

1. **Test Results Tab:**
   * This tab shows the overall status of the test execution after running the script.
   * It includes:
     + The **Pass/Fail status** of each test step.
     + **Error messages** and screenshots for failed steps.
     + A **summary** of the test run and detailed logs for each step.
   * **Usage in Debugging:**
     + If a test fails, you can look at the **Test Results Tab** to identify the failed step.
     + You can analyze the failure by checking the **error message** and reviewing the **log details**.
     + The **screenshot** taken at the point of failure can help identify issues.
2. **Run-Time Data Table:**
   * This table holds the **data used during test execution** for parameterized testing.
   * It allows you to **view and modify** the values of the test data while the test is running.
   * The table can be accessed and updated during runtime, and it is often used in conjunction with **data-driven testing** to feed different sets of inputs.
   * **Usage in Debugging:**
     + If a test fails due to incorrect data, you can check the **Run-Time Data Table** to ensure the correct data was passed to the test steps.
     + By reviewing the table, you can ensure that the correct parameters were being used at the time of failure, especially in data-driven tests.

**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.**

Here’s an example script that generates a custom report in UFT after executing a test case:

' Custom Report Script in UFT

' Initialize variables for report

Dim objFSO, objFile

Dim reportPath, reportContent

reportPath = "C:\TestResults\CustomReport.html"

reportContent = ""

' Start generating the report content

reportContent = "<html><body><h2>Test Execution Report</h2>"

reportContent = reportContent & "<table border='1'><tr><th>Step</th><th>Status</th><th>Message</th></tr>"

' Example test steps (replace with actual UFT test steps)

Dim step1Status, step2Status

step1Status = "Pass" ' Example status

step2Status = "Fail" ' Example status

' Step 1 - Example

reportContent = reportContent & "<tr><td>Step 1: Open Browser</td><td>" & step1Status & "</td><td>Browser opened successfully</td></tr>"

' Step 2 - Example

reportContent = reportContent & "<tr><td>Step 2: Click Submit Button</td><td>" & step2Status & "</td><td>Failed to click button (Object not found)</td></tr>"

' Add more steps as needed

' Close the table and HTML tags

reportContent = reportContent & "</table></body></html>"

' Create the file system object to write the file

Set objFSO = CreateObject("Scripting.FileSystemObject")

' Check if directory exists, if not, create it

If Not objFSO.FolderExists("C:\TestResults") Then

objFSO.CreateFolder("C:\TestResults")

End If

' Create and open the report file for writing

Set objFile = objFSO.CreateTextFile(reportPath, True)

objFile.WriteLine(reportContent)

' Close the file

objFile.Close

' Output message indicating report is saved

MsgBox "Test report saved to " & reportPath

**Explanation:**

* This script generates an HTML report after running a test case. It includes:
  + Test step name
  + The pass/fail status of each step
  + A relevant message describing what happened during each test step.
* It uses **FileSystemObject** (objFSO) to create and write to an HTML file.
* The file is saved at C:\TestResults\CustomReport.html, but you can change the path as needed.
* You can modify the status and messages dynamically based on the test execution outcome.