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?

  Answer:

  What is UFT (Unified Functional Testing)
  UFT is a commercial test automation tool developed by Micro Focus, used for functional and regression testing of software applications. It supports testing of desktop, web, and mobile applications and allows both keyword-driven and scripting approaches using VBScript.

  Differences between UFT, Selenium, and QTP

  1. **Technology Support**
     - UFT: Supports a wide range of applications including desktop, web, and mobile.
     - Selenium: Focuses only on web application testing.
     - QTP: UFT is actually the newer version of QTP with more features.
  2. **Scripting Language**
     - UFT/QTP: Uses VBScript.
     - Selenium: Supports multiple languages like Java, Python, C#, etc.
  3. **Ease of Use**
     - UFT: Has a user-friendly GUI with keyword-driven testing and record/playback features.
     - Selenium: Requires more coding and technical knowledge.
  4. **Licensing**
     - UFT: Paid commercial tool.
     - Selenium: Open-source and free.
  5. **Integration**
     - UFT: Integrates well with Micro Focus ALM, Jenkins, and other test management tools.
     - Selenium: Integrates with various third-party tools but requires more setup.

  UFT is best suited for enterprise-level applications where broader technology coverage and integrated tools are required, while Selenium is preferred for flexible and open-source web testing.

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

Answer:

Key Features of UFT (Unified Functional Testing)

1. Keyword and Script-Based Testing
Supports both keyword-driven and scripting (VBScript) approaches, making it suitable for both technical and non-technical users.

2. Wide Technology Support
Can test web, desktop, mobile, API, and legacy applications across various platforms.

3. Object Repository
Stores properties of UI elements, making it easy to manage and reuse objects in test scripts.

4. Integration with ALM and CI Tools
Seamlessly integrates with Micro Focus ALM (Application Lifecycle Management), Jenkins, and other tools for test management and continuous testing.

5. Record and Playback
Provides an easy way to create test scripts by recording user actions, which can be replayed for automation.

6. Data-Driven Testing
Supports testing with multiple data sets using external files like Excel, enabling thorough validation.

7. Error Handling and Debugging Tools
Includes built-in features to handle exceptions, debug scripts, and add checkpoints to verify expected results.

How UFT Supports Functional, Regression, and GUI Testing

Functional Testing
UFT checks if the application behaves as expected by testing each function or feature (like login, form submission) independently or in sequence.

Regression Testing
Automatically re-runs previously created test scripts after code changes to ensure existing functionality still works as intended.

GUI Testing
Interacts with the user interface (buttons, forms, menus) to verify layout, behavior, and usability through object-based testing.

UFT provides a complete testing solution by combining functional correctness, UI verification, and regression assurance in one tool.

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

Answer:

Types of Objects Recognized by UFT and Examples

1. Standard GUI Objects
These are common user interface elements found in desktop and web applications.
Examples:

- Button (e.g., "Submit")
- Edit box (e.g., username field)
- Checkbox (e.g., "Remember Me")
- Radio button (e.g., "Male" or "Female")
- Combo box/drop-down (e.g., "Select Country")

2. Web Objects
Used when testing web-based applications.
Examples:

- WebEdit (e.g., text input in a login form)
- WebButton (e.g., "Login" button)
- WebLink (e.g., "Forgot Password?" link)
- WebList (e.g., a drop-down list on a web page)
- WebTable (e.g., data table displaying search results)

3. ActiveX Controls
Third-party or custom controls embedded in applications.
Examples:

- Calendar control
- Slider control
- Rich text editor

4. .NET Objects
Used in applications developed using Microsoft .NET framework.
Examples:

- SwfButton
- SwfEdit
- SwfList
  These are specific to .NET environments and often begin with "Swf"

5. Java Objects
Recognized in Java-based applications.
Examples:

- JavaButton
- JavaEdit
- JavaTable
- JavaTree

6. SAP Objects
Used for testing SAP GUI applications.
Examples:

- SAPGuiEdit
- SAPGuiButton
- SAPGuiCheckBox

7. Custom or User-Defined Objects
When UFT cannot recognize a control using its standard object model, testers can define their own object types using virtual objects or descriptive programming.
Example:

- A custom-drawn button in a gaming application

By recognizing a wide range of object types, UFT can be used to test many different applications and technologies effectively.

---

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.

Answer:

Creating a Simple Test in UFT to Open Notepad, Type Text, and Save the File

Steps to Record and Run the Test in UFT

1. Launch UFT
Open Unified Functional Testing on your computer.

2. Create a New Test

- Go to `File > New > Test`
- Select "GUI Test" and name your test
- Click "Create"

3. Start Recording

- Click on the **Record** button (usually red circle on the toolbar)
- In the Record and Run Settings, choose **Record and run test on any open Windows-based application**
- Click OK

4. Perform the Actions to Be Recorded
UFT will now capture your actions:

- Press `Windows + R`, type `notepad`, and hit Enter to open Notepad
- In Notepad, type a message like: `This is a test message from UFT.`
- Go to `File > Save As`
- In the Save dialog, enter a file name like `UFT_Test.txt`
- Choose a location (like Desktop), and click **Save**

5. Stop Recording

- Click the **Stop** button in UFT

6. View the Script
UFT will display the generated VBScript in the Expert View. It might look like this:

```
SystemUtil.Run "notepad.exe"
Window("Notepad").WinEditor("Edit").Set "This is a test message from
UFT."
Window("Notepad").WinMenu("Menu").Select "File;Save As..."
Window("Save As").WinEdit("File name:").Set "UFT_Test.txt"
Window("Save As").WinButton("Save").Click
Window("Notepad").Close
```

```
Window("Notepad").Dialog("Notepad").WinButton("Don't Save").Click 'if
prompted
```

7. Run the Test

- Click the **Run** button
- Choose a result location if prompted
- Watch UFT open Notepad, type the message, and save the file

This basic test shows how to automate simple tasks on desktop applications using UFT's recording feature.

- **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.

Answer:

Here's a simple UFT script to open a browser, navigate to **www.google.com**, enter a search term, and perform a Google search using **Descriptive Programming**:

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

' Step 2: Wait for the page to load
Browser("title:=Google.*").Page("title:=Google.*").Sync

' Step 3: Enter search text in the search box
Browser("title:=Google.*").Page("title:=Google.*").WebEdit("name:=q").S
et "UFT automation testing"

' Step 4: Click the Google Search button
Browser("title:=Google.*").Page("title:=Google.*").WebButton("name:=Goo
gle Search").Click
```

Note:

- Make sure Chrome is properly configured and UFT supports it (with the **UFT Web Add-in** enabled).
- You may need to use **Object Spy** to confirm the properties of the search box and button if this script doesn't match your environment exactly.
- For Chrome support, ensure **UFT Agent extension** is installed and enabled in Chrome.

---

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."

  Answer:

  What is an Object Repository in UFT?

  An **Object Repository (OR)** in UFT is a storage location where UFT stores information (properties and values) about the GUI objects it interacts with during a test. It helps UFT identify and interact with elements like buttons, text boxes, and links in an application.

  Types of Object Repositories in UFT

  1. Local Object Repository (LOR)

  - Stores objects specific to a single action in a test
  - Automatically created when you record a test
  - Saved within the test itself
  - Easier to manage for small or one-time tests
  - Not reusable across multiple tests

  Example:
  If you record a test that clicks a login button, the button's properties are stored only in that test's action.

  2. Shared Object Repository (SOR)

  - Stored as a separate `.tsr` file
  - Can be used by multiple tests or actions
  - Ideal for large projects where multiple tests interact with the same objects
  - Promotes reusability and consistency

  Example:
  If multiple test scripts use the same "Login" page, a shared repository allows all scripts to reference the same set of objects.

  Key Differences

Feature	Local Object Repository	Shared Object Repository
Scope	Specific to one action/test	Can be used across multiple tests
Reusability	Not reusable	Reusable
Storage	Inside the test	External `.tsr` file

Feature	Local Object Repository	Shared Object Repository
Maintenance	Hard to manage for many tests	Easier to maintain in large projects
Best Used For	Small or quick tests	Large, modular test frameworks

Both types help UFT recognize UI elements, but shared repositories are more efficient for collaborative and long-term projects.

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

Answer:

Object Identification in UFT

Object Identification is the process UFT uses to recognize and interact with UI elements (objects) like buttons, text fields, or drop-downs in the application being tested. UFT must correctly identify these objects to perform actions like clicking or entering text.

How UFT Recognizes Objects

1. **Property-Based Identification**
   UFT identifies objects using a set of properties (like name, id, class, or value).
   Example: A login button may be identified by its `html tag = button` and `name = Login`.
2. **Object Repository**
   UFT stores these object properties in the **Object Repository** (local or shared). During test execution, it compares the stored properties with those on the application screen to find a match.
3. **Ordinal Identifier**
   If multiple objects have similar properties, UFT uses **ordinal identifiers** like index, location, or creation time to distinguish between them.
4. **Smart Identification (Fallback Method)**
   If UFT cannot find an object using normal property values, it applies **Smart Identification**. This checks a set of **base filter properties** and **optional filter properties** to find the best match.

Example

For a "Submit" button, UFT might store:

- Class: `WebButton`
- Name: `Submit`

- HTML tag: `INPUT`

If multiple buttons match, it may use the **index** or **position on screen** to identify the correct one.

---

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.

  Answer:

  What Are Checkpoints in UFT?
  Checkpoints in UFT are verification points that compare the actual behavior or content of the application under test with the expected results during test execution. They help validate whether the application is working correctly.

  Types of Checkpoints Include

  - **Standard Checkpoint** – Verifies properties of objects (e.g., enabled, visible).
  - **Text Checkpoint** – Checks if specific text appears on the screen or in an object.
  - **Bitmap Checkpoint** – Verifies the appearance of images.
  - **Database Checkpoint** – Validates data in databases.
  - **XML Checkpoint** – Checks XML content.

  Script to Add a Text Checkpoint in UFT

  Here's a sample script to verify that a specific text (e.g., "Welcome to Google") appears on a web page using a **Text Checkpoint**:

```
' Open Chrome and navigate to Google
SystemUtil.Run "chrome.exe", "https://www.google.com"

' Wait for page to load
Browser("title:=Google").Page("title:=Google").Sync

' Add a Text Checkpoint to verify "Google offered in"
Browser("title:=Google").Page("title:=Google").Check
CheckPoint("GoogleText")

' Note: "GoogleText" must be created using the UI by adding a Text
Checkpoint from the menu
```

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

Answer:

Difference Between Standard Checkpoints and Database Checkpoints in UFT

1. Standard Checkpoints

- **Purpose**: Used to verify properties of objects in the application, such as buttons, text fields, labels, or web elements.
- **What it checks**: Object properties like text, enabled/disabled status, tooltips, or size.
- **When to use**: When you want to validate that a UI element displays correct information or behaves as expected.

Example:
You want to check that the "Login" button is visible and enabled after the user enters valid credentials.

```
Browser("LoginPage").Page("LoginPage").WebButton("Login").Check
CheckPoint("LoginButtonEnabled")
```

2. Database Checkpoints

- **Purpose**: Used to validate the data in a database (e.g., SQL, Oracle, MS Access).
- **What it checks**: Values in rows/columns of a database table, comparing actual data with expected values.
- **When to use**: When you need to verify that a user action correctly updates the database.

Example:
After submitting a registration form, you want to check that the user's email was stored correctly in the database.

```
' Inserted via Insert > Checkpoint > Database Checkpoint
' UFT connects to DB using a connection string and validates query
result
```

Summary

Feature	Standard Checkpoint	Database Checkpoint
Validates	UI object properties	Data stored in database tables
Usage example	Checking if a button is enabled	Checking if user data was inserted correctly

Feature	Standard Checkpoint	Database Checkpoint
Added from	GUI object on screen	SQL query via database connection

We use **Standard Checkpoints** for front-end validation and **Database Checkpoints** for back-end data integrity.

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

Answer:

Handling Dynamic Objects in UFT

Dynamic objects have properties (like name, ID, or text) that change each time the application runs. UFT may fail to recognize them if it relies on fixed values. To handle these, you can use **Descriptive Programming** or **Regular Expressions**.

1. Descriptive Programming
Instead of using the object repository, define the object directly in the script using property-value pairs.

Example – Handling a dynamic button where the text changes (e.g., "Save - 1", "Save - 2", etc.):

```
' Use partial matching with regular expression
Set btnSave =
Browser("title:=MyApp").Page("title:=MyApp").WebButton("text:=Save.*")

' Click the dynamic Save button
btnSave.Click
```

2. Regular Expressions in Object Repository
If using the object repository, enable regular expressions in object properties.

Steps:

- Open the object in the repository
- Set the `text` or `name` property to a pattern like `Save.*`
- Check the "Regular Expression" box

3. Using Ordinal Identifiers
If objects have similar properties, you can use index or location to identify the correct one.

Example:

```
Browser("title:=MyApp").Page("title:=MyApp").WebButton("index:=1").Clic
k
```

These methods allow UFT to reliably identify and interact with dynamic elements.

---

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).

Answer:

What is Parameterization in UFT

Parameterization is the process of replacing hardcoded values in a test (like usernames or passwords) with variables that can take different values during test execution. It allows tests to run multiple times with different input data.

Why Parameterization is Important

- Makes tests reusable for multiple data sets
- Helps in data-driven testing
- Saves time by avoiding duplicate test scripts
- Increases test coverage with varied inputs

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

Scenario: Automate login for multiple users using a data table

Steps:

1. Open UFT and record or write this test:

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

```
Browser("LoginPage").Page("LoginPage").WebEdit("name:=password").Set
"admin123"
Browser("LoginPage").Page("LoginPage").WebButton("name:=Login").Click
```

2. Replace hardcoded values with parameters from the **Data Table**:

```
Browser("LoginPage").Page("LoginPage").WebEdit("name:=username").Set
DataTable("Username", dtGlobalSheet)
Browser("LoginPage").Page("LoginPage").WebEdit("name:=password").Set
DataTable("Password", dtGlobalSheet)
Browser("LoginPage").Page("LoginPage").WebButton("name:=Login").Click
```

3. Go to **Data > Data Table**
4. Add columns named **Username** and **Password**
5. Enter different sets of credentials in each row
6. UFT will automatically loop through each row of data during test execution

This way, the test can log in with multiple user accounts in one run.

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

Answer:

UFT Test Using Input Parameters from an Excel File for Login

To create a test that reads **username** and **password** from an external Excel file and performs a login, follow these steps:

1. Prepare the Excel File

- Open Excel and enter the following:

Username	Password
user1	pass123
user2	pass456

- Save the file as `LoginData.xlsx` (e.g., at `C:\TestData\LoginData.xlsx`)

2. UFT Script to Use Excel Data

```
' Create Excel objects
Set objExcel = CreateObject("Excel.Application")
```

```
Set objWorkbook = objExcel.Workbooks.Open("C:\TestData\LoginData.xlsx")
Set objSheet = objWorkbook.Sheets(1)

' Find number of rows
rowCount = objSheet.UsedRange.Rows.Count

' Loop through Excel rows (starting from row 2 assuming row 1 is
header)
For i = 2 To rowCount
    username = objSheet.Cells(i, 1).Value
    password = objSheet.Cells(i, 2).Value

    ' Perform login
    SystemUtil.Run "chrome.exe", "https://example-login-page.com" '
Replace with actual login URL

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

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

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

    ' Add wait or verification steps here if needed
    Browser("title:=.*").Close
Next

' Close Excel
objWorkbook.Close False
objExcel.Quit
Set objSheet = Nothing
Set objWorkbook = Nothing
Set objExcel = Nothing
```

3. Notes

- Use **Object Spy** to get the correct properties (`name`, `html id`, etc.) for the username, password fields, and login button.
- The script will open the browser, log in with each set of credentials, and close the browser after each attempt.
- Add validations to check for successful login if needed.

This test automates login using multiple inputs from Excel, enabling **data-driven testing** outside the default UFT data table.

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

Answer:

Types of Parameters in UFT and Their Use

1. Test Parameters

- These are defined at the test level and are used to pass values into the entire test.
- Useful when running the same test with different inputs from an external controller like Quality Center or calling test.

Example:
A login test that accepts `Username` and `Password` as test parameters.

```
username = Parameter("Username")
password = Parameter("Password")
```

2. Action Parameters

- Defined at the action level to pass data into or out of specific actions.
- Useful when dividing a test into multiple actions and sharing values between them.

Example:
Action1 has an input parameter `UserID`, which is used inside the action:

```
msgbox "UserID is: " & Parameter("UserID")
```

3. Data Table Parameters

- Values are taken from the **Data Table** (Global or Local sheet).
- Used for **data-driven testing** to run the same test multiple times with different sets of data.

Example:
DataTable has columns `Username` and `Password`:

```
username = DataTable("Username", dtGlobalSheet)
password = DataTable("Password", dtGlobalSheet)
```

4. Environment Variables

- These are global variables that can be defined internally in UFT or loaded from external `.xml` files.
- Useful for storing values like URLs, credentials, or file paths.

Example:

```
url = Environment("AppURL")
```

Summary of Uses

Parameter Type	Scope	Example Use
Test Parameter	Entire test	Pass login credentials from external run
Action Parameter	Specific action only	Share user input between actions
Data Table Param	One or multiple iterations	Run test with multiple user records
Environment Var	Global configuration	Set app URL or browser type

Each parameter type helps improve reusability, flexibility, and scalability of test automation in UFT.

---

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.

Answer:

What is an Action in UFT
An **action** in UFT is a modular block of code within a test that performs a specific task. Actions help break down a test into smaller, manageable parts—each representing a distinct step or function (like login, search, or logout).

There are two types:

1. **Reusable Action** – Can be called from multiple tests or actions
2. **Non-reusable Action** – Used only within the test it was created in

How Actions Help Organize Test Scripts

- Promote **modularity** and **reusability**
- Make scripts easier to **maintain and update**
- Allow **data sharing** through input/output parameters
- Improve **readability** by separating test logic into logical units

Example: Reusable Action for Login

Step 1: Create a Reusable Action

- Go to `Edit > Action > Action Properties`

- Check **Reusable Action** box

Step 2: Add Parameters to the Action

- Input Parameters: `Username`, `Password`

Step 3: Script Inside the Action

```
' Get values from input parameters
user = Parameter("Username")
pass = Parameter("Password")

' Perform login
Browser("title:=.*").Page("title:=.*").WebEdit("name:=username").Set
user
Browser("title:=.*").Page("title:=.*").WebEdit("name:=password").Set
pass
Browser("title:=.*").Page("title:=.*").WebButton("name:=Login").Click
```

Step 4: Call the Action from Another Test or Action

```
RunAction "LoginAction", oneIteration, "user1", "pass123"
```

This setup makes the login process reusable across different tests without rewriting the login logic every time.

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

Answer:

Function Libraries in UFT

A **Function Library** in UFT is a separate file (usually with a `.vbs` or `.qfl` extension) that contains user-defined functions, reusable code, or business logic that can be used across multiple test scripts.

Purpose of Function Libraries

- Promote **code reusability**
- Keep the test script clean and modular
- Centralize logic to make maintenance easier
- Enable multiple tests to share common functions

How to Create a Function Library

1. In UFT, go to `File > New > Function Library`
2. Write your reusable functions in VBScript
3. Save the file with a `.vbs` or `.qfl` extension (e.g., `LoginLibrary.vbs`)

Example Function (Login)

```
Function Login(username, password)

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

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

Browser("title:=.*").Page("title:=.*").WebButton("name:=Login").Click
End Function
```

How to Associate a Function Library with Your Test

Option 1: Manually Associate

- Go to `File > Settings > Resources` tab
- Click **+** to browse and add the `.vbs` or `.qfl` file
- Click **OK** to associate it with your test

Option 2: Use LoadFunctionLibrary (at runtime)

```
LoadFunctionLibrary "C:\Path\To\LoginLibrary.vbs"
```

Calling the Function in Your Test

```
Login "admin", "admin123"
```

Function libraries make your automation framework flexible, scalable, and easy to maintain.

- **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.

Answer:

Step 1: Create the Function in a Function Library

Open UFT, go to `File > New > Function Library` and add the following code:

```
' FunctionLibrary.vbs

Function AddNumbers(a, b)
    AddNumbers = a + b
End Function
```

Save the file as `FunctionLibrary.vbs` at a known path (e.g., `C:\UFT\FunctionLibrary.vbs`).

Step 2: Associate the Function Library with Your Test

Option 1: Load at runtime in your test script

```
LoadFunctionLibrary "C:\UFT\FunctionLibrary.vbs"
```

Option 2: Add it via File > Settings > Resources tab

Step 3: Call the Function from Your Test Script

```
' Call the function and display the result
result = AddNumbers(10, 20)
MsgBox "The sum is: " & result
```

This simple setup demonstrates how to define and reuse a custom function using a UFT function library.

---

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).

  Answer:

  What is Descriptive Programming in UFT

  Descriptive Programming (DP) is a method in UFT where you describe objects directly in the script using their properties, instead of relying on the Object Repository. This is useful when:

  - You don’t want to use or update the Object Repository
  - Objects are dynamic and created during runtime
  - You're working with many similar objects

- You want to create quick, lightweight scripts

When to Use Descriptive Programming

- For dynamic object identification
- In data-driven or reusable function libraries
- For working with objects not present in the Object Repository
- In keyword-driven frameworks where code is more generic

Example: Click a "Submit" Button Using Descriptive Programming

```
' Launch browser and open the webpage
SystemUtil.Run "chrome.exe", "https://example.com"

' Wait for page to load
Browser("title:=.*").Page("title:=.*").Sync

' Click the Submit button using descriptive programming
Browser("title:=.*").Page("title:=.*").WebButton("type:=submit",
"name:=Submit").Click
```

In this example, the button is described by its HTML properties:

- `type:=submit`
- `name:=Submit`

You can adjust or add properties like `html id`, `class`, or `value` depending on the actual page. Use **Object Spy** to inspect the correct properties of the element.

- **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.).

Answer:

Syntax for Descriptive Programming in UFT

Descriptive Programming allows you to describe an object using a set of **property–value** pairs directly in the script. This is done using:

```
ObjectType("property1:=value1", "property2:=value2")
```

You can nest this inside browser and page objects similarly:

```
Browser("title:=.*").Page("title:=.*").WebElement("html tag:=A",
"innertext:=Click Here").Click
```

Example Script: Interact with a Web Link Using Descriptive Programming

```
' Launch the browser and navigate to the webpage
SystemUtil.Run "chrome.exe", "https://example.com"

' Wait for the page to load
Browser("title:=.*").Page("title:=.*").Sync

' Click a hyperlink with the link text "Contact Us"
Browser("title:=.*").Page("title:=.*").WebElement("html tag:=A",
"innertext:=Contact Us").Click
```

Explanation of Properties Used

- `"html tag:=A"` — identifies an anchor tag (a link)
- `"innertext:=Contact Us"` — matches the visible link text

You can modify the properties to match any web element, such as buttons (`WebButton`), text fields (`WebEdit`), etc., using appropriate identifiers like `name`, `html id`, `class`, or `value`.

This approach is especially helpful for dynamic elements or when the Object Repository is not practical.

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

Answer:

Handling Dynamic Objects with Descriptive Programming in UFT

UFT handles dynamic objects by using **Descriptive Programming with Regular Expressions** or **partial property matching**. This is helpful when object properties (like name, ID, or text) change dynamically at runtime but follow a predictable pattern.

Techniques Used

1. **Regular Expressions** in property values
2. Using **wildcards or partial matches** (e.g., `.*` for any characters)
3. Minimal, **stable property sets** to reduce dependency on dynamic values

Example: Click a Dynamic Link Using Descriptive Programming

Suppose a webpage contains links like:

- "Order #12345"
- "Order #67890"
  And the number changes each time.

You can handle this with a regular expression in `innertext`:

```
' Click a link with text like "Order #<number>"
Browser("title:=.*").Page("title:=.*").WebElement("html tag:=A",
"innertext:=Order #\d+").Click
```

Here:

- `html tag:=A` — identifies it as a link
- `innertext:=Order #\d+` — uses a regex to match "Order #" followed by digits

Example: Click a Dynamic Button with Changing ID

If a button has an ID like `btnSubmit_123`, `btnSubmit_456`, etc.

```
Browser("title:=.*").Page("title:=.*").WebButton("html
id:=btnSubmit_.*", "type:=submit").Click
```

- `html id:=btnSubmit_.*` — matches any ID that starts with `btnSubmit_`
- `type:=submit` — confirms it's a submit button

Note:
To use regular expressions in Descriptive Programming:

- Just include the regex pattern in the value (e.g., `.*`, `\d+`)
- No need to explicitly enable regex — UFT treats these patterns as regex by default in DP

This method makes your tests more robust and flexible when working with UI elements that change dynamically.

---

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?

Answer:

Why Synchronization is Important in UFT

Synchronization is essential in UFT to ensure that test steps execute only when the application is ready. Applications may take time to load elements due to slow internet, server response, or animations. Without synchronization, UFT may try to interact with objects before they are fully available, causing errors or test failures.

Synchronization Techniques in UFT

1. Wait Property Method
Waits until a specific object property (like "visible" or "enabled") becomes true or a timeout occurs.

```
Browser("title:=.*").Page("title:=.*").WebButton("name:=Login").WaitPro
perty "visible", True, 10000
```

2. Sync Method
Used to wait for a browser page to load completely.

```
Browser("title:=.*").Page("title:=.*").Sync
```

3. Exist Method
Checks if an object exists on the screen and waits for a specified time.

```
If
Browser("title:=.*").Page("title:=.*").WebElement("innertext:=Welcome")
.Exist(10) Then
    MsgBox "Element is available"
End If
```

4. Wait Statement
Pauses the test execution for a fixed number of seconds. It is simple but not recommended as a primary method because it's not dynamic.

```
Wait 5 'Waits for 5 seconds
```

5. Smart Synchronization
UFT automatically adds synchronization during recording for common actions like clicking or setting text, but it's limited and sometimes needs to be adjusted manually.

6. Setting Global Synchronization Timeout
From UFT: `File > Settings > Run`, set **Object Synchronization Timeout** (default is 20 seconds). This tells UFT how long to wait for any object.

Using proper synchronization techniques ensures your automation is stable, reliable, and able to handle varying load times.

- **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.

  Answer:

  Here's a simple UFT script that demonstrates using both the **`Sync`** method and **`Wait`** method to ensure a webpage is fully loaded before interacting with a button:

```
' Launch browser and navigate to the URL
SystemUtil.Run "chrome.exe", "https://example.com"

' Sync method waits for the browser page to fully load
Browser("title:=.*").Page("title:=.*").Sync

' Optional: wait for 3 seconds to handle any slow UI rendering
Wait 3

' Click a button (e.g., Submit) using Descriptive Programming
Browser("title:=.*").Page("title:=.*").WebButton("name:=Submit").Click
```

  Explanation:

  - `Sync`: Waits until the browser and page finish loading.
  - `Wait 3`: Pauses execution for 3 seconds to allow any additional elements (like animations or scripts) to finish.
  - `WebButton`: Locates and clicks the "Submit" button after ensuring the page is ready.

  You can adjust the `Wait` time based on your application's response time, but combining `Sync` and `WaitProperty` is usually more efficient than using only `Wait`.

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

  Answer:

  To handle synchronization issues in UFT when testing a slow application or pages with dynamic content, use the following techniques to ensure your script waits intelligently and reliably:

1. Use `Sync` Method
Waits for the browser to finish loading the page. Use this immediately after navigation.

```
Browser("title:=.*").Page("title:=.*").Sync
```

2. Use `WaitProperty` Method
Waits until a specific object property (like `visible`, `enabled`, or `exist`) becomes true within a timeout.

```
Browser("title:=.*").Page("title:=.*").WebButton("name:=Submit").WaitPr
operty "visible", True, 10000
```

3. Use `Exist` Method
Checks if an object exists and allows you to wait for a specific duration.

```
If
Browser("title:=.*").Page("title:=.*").WebElement("innertext:=Loading
complete").Exist(15) Then
    ' Proceed with next steps
End If
```

4. Set Global Synchronization Timeout
Go to `File > Settings > Run` and increase **Object Synchronization Timeout** to allow more time for objects to appear.

5. Avoid Using Fixed `Wait` Unless Necessary
Using `Wait` adds a static pause. It’s simple but not efficient. Use it only when there's no better alternative.

```
Wait 5 ' Waits for 5 seconds regardless of page readiness
```

6. Use Smart Wait (UFT 15 and above)
Enable Smart Wait in UFT settings to automatically wait for dynamic changes in DOM (like AJAX or JavaScript updates).

7. Combine Techniques
Use a combination of `Sync`, `WaitProperty`, and `Exist` for better control and to handle different application behaviors.

Proper synchronization ensures stable and reliable test execution even with slow-loading or dynamic content.

---

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?

  Answer:

  To handle unexpected pop-ups or alerts during test execution in UFT, you can use **exception handling** techniques such as `On Error Resume Next`, `Recovery Scenarios`, or checking for pop-ups using object existence.

  1. Using VBScript Error Handling (`On Error Resume Next`)

  This prevents the script from crashing due to runtime errors and lets you handle exceptions gracefully.

```
On Error Resume Next

' Try to click a button
Browser("title:=.*").Page("title:=.*").WebButton("name:=Submit").Click

If Err.Number <> 0 Then
    MsgBox "An error occurred: " & Err.Description
    Err.Clear
End If

On Error GoTo 0
```

  2. Handling Pop-ups or Alerts with `Exist` Method

  Check if a pop-up window or alert exists before interacting with it.

```
If Browser("title:=.*").Dialog("text:=.*Error.*").Exist(5) Then

Browser("title:=.*").Dialog("text:=.*Error.*").WinButton("text:=OK").Cl
ick
End If
```

  3. Using Recovery Scenarios (UFT Feature)

  To automatically handle unexpected events like pop-ups:

  - Go to `Resources > Recovery Scenario Manager`
  - Create a new recovery scenario
  - Trigger event: Choose **popup window**, **object state**, or **test run error**
  - Recovery operation: Choose action (e.g., press button, close window, call function)
  - Post-recovery: Choose whether to repeat step, continue, or stop

Then, associate the recovery file with your test using `File > Settings > Recovery`.

4. Custom Function for Handling Alerts

Create a reusable function:

```
Function HandleAlert()
    If Browser("title:=.*").Dialog("text:=.*").Exist(3) Then

Browser("title:=.*").Dialog("text:=.*").WinButton("text:=OK").Click
    End If
End Function
```

Call `HandleAlert` after actions that might trigger alerts.

---

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?

Answer:

How UFT Generates Test Results

When a test is executed in UFT, it automatically generates a **Test Result Report** that provides detailed information about the execution. The report includes:

- Overall test status (Passed, Failed, Warning)
- Execution time and duration
- Step-by-step details of each action and operation
- Screenshots (for checkpoints or failures, if enabled)
- Errors and messages encountered during execution

How to View and Analyze Test Results

1. After Test Execution

- UFT automatically opens the **Test Results Viewer** after the test completes.
- You can also access it anytime by going to:
  `Run > Test Results` or pressing `Ctrl + Alt + R`.

2. Structure of the Test Results Viewer

- **Summary Tab**: Shows overall test outcome, execution time, total steps, passed/failed steps.
- **Results Tree**: Displays a step-by-step breakdown of the actions performed.
- **Details Pane**: Provides detailed info for each step (e.g., input values, expected vs. actual results).
- **Screen Captures**: If screen capture is enabled, you can view the screenshots of passed or failed steps.
- **Errors/Warnings**: Describes errors with suggestions for debugging.

3. Exporting Test Results

- You can export results as **HTML** or **PDF** using the Export option in the viewer.
- Optionally, results can be integrated with tools like **ALM (Application Lifecycle Management)** or CI pipelines for centralized reporting.

4. Customizing Results

- In `Tools > Options > Run Sessions`, you can configure:
  - Whether to take screenshots on errors
  - The level of detail in the report
  - Log tracking options

Conclusion
UFT's result reporting helps testers quickly identify what passed, what failed, and why—making it easier to analyze issues and improve test reliability.

- **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?

Answer:

Difference Between "Test Results" Tab and "Run-Time Data Table" in UFT

1. Test Results Tab

- Displays the **step-by-step outcome** of the executed test.
- Shows which steps passed, failed, or were skipped.
- Includes **messages, errors**, screenshots (if enabled), and detailed info for each step.
- Helps understand **what happened** during test execution.

Usage in Debugging:

- Pinpoints **exactly which step failed**
- Displays **error messages and expected vs actual** values
- Useful for identifying logical or object recognition errors

2. Run-Time Data Table

- A **copy of the data table** used during test execution
- Shows the **actual data values** used at run time (input/output parameters)
- Updated if script modifies the data table during the test
- Can be accessed from `File > Last Run Results > Data Table` or in `Test Results Viewer`

Usage in Debugging:

- Helps check **what data was passed** to a step when it failed
- Useful for **data-driven tests** to verify values that caused the failure
- Detects issues related to missing, invalid, or incorrect test data

Example Debug Scenario

If a login test fails at the step where the username is entered:

- Check **Test Results tab** for the error message (e.g., "Object not found" or "Invalid credentials")
- Then open the **Run-Time Data Table** to see what username and password were used during that iteration

Using both views together helps you identify **whether the failure was due to code logic, object recognition, or test data**.

- **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.

Answer:

Here's a sample UFT script that creates a **custom report** by writing test results (step name, status, and message) to a plain text file:

Step-by-Step Script: Custom Report Generation

```
' Define report file path
reportPath = "C:\UFT_Custom_Report.txt"
```

```
' Create or overwrite the report file
Set fso = CreateObject("Scripting.FileSystemObject")
Set reportFile = fso.CreateTextFile(reportPath, True)

' Function to log test step results
Sub LogStep(stepName, status, message)
    reportFile.WriteLine "Step: " & stepName
    reportFile.WriteLine "Status: " & status
    reportFile.WriteLine "Message: " & message
    reportFile.WriteLine "" ' Blank line between steps
End Sub

' --- Sample Test Case Execution with Logging ---

' Example: Open browser and navigate
On Error Resume Next
SystemUtil.Run "chrome.exe", "https://example.com"
If Err.Number = 0 Then
    LogStep "Open Browser", "Pass", "Browser launched successfully."
Else
    LogStep "Open Browser", "Fail", "Failed to launch browser: " &
Err.Description
    Err.Clear
End If

Wait 3

' Example: Interact with a button using descriptive programming
On Error Resume Next
Set btn =
Browser("title:=.*").Page("title:=.*").WebButton("name:=Submit")
If btn.Exist(5) Then
    btn.Click
    LogStep "Click Submit Button", "Pass", "Submit button clicked."
Else
    LogStep "Click Submit Button", "Fail", "Submit button not found."
End If
On Error GoTo 0

' --- End of Test ---
reportFile.WriteLine "Test Completed on: " & Now
reportFile.Close
Set reportFile = Nothing
Set fso = Nothing

MsgBox "Custom test report generated at: " & reportPath
```

Output:

A text file named `UFT_Custom_Report.txt` will be created at `C:\`, containing a log like:

```
Step: Open Browser
Status: Pass
Message: Browser launched successfully.
```

```
Step: Click Submit Button
Status: Pass
Message: Submit button clicked.

Test Completed on: 6/15/2025 3:30:22 PM
```

You Can Enhance This Further:

- Use Excel or HTML format instead of text
- Add timestamps per step
- Capture screenshots and add their paths to the log

Let me know if you'd like this in Excel or HTML format next.
