1. Briefly Describe about Current Company?

**Ans:** Just tell about HCL, like it is CMM level 3 company something like.

1. Briefly Describe about your Profile?

**Ans:** Just explain about your Profile including current roles and responsibilities.

1. Briefly describe about the processes you are following?

**Ans**: Describe about the processes that you are following like how you get requirements, how do you design Test scenarios and test cases, How do u maintain testing related documentation like traceability matrix, TC’s, Test scenario docs and other project planning sheets. Describe about some new things like Scrum, Agile process and agile tools like rally etc. Answer everything in such a way that you are satisfying the standard process like SDLC, STLC, and BLC etc in your project.

1. Type of Recovery Scenarios and its Navigations in QTP?

**Ans:** While executing your scripts you may get some UNEXPECTED/UNPREDICTABLE errors. (like printer out of paper). To "recover" the test (and continue running) from these unexpected errors you use Recovery Scenarios.

**Type of Recovery Scenarios**

* **Trigger Event.** The event that interrupts your run session. For example, a window that may pop up on screen, or a QTP run error.
* **Recovery Operations.** The operations to perform to enable QTP to continue running the test after the trigger event interrupts the run session. For example, clicking an **OK** button in a pop-up window, or restarting Microsoft Windows.
* **Post-Recovery Test Run Option.** The instructions on how QTP should precede after the recovery operations have been performed, and from which point in the test QTP should continue, if at all. For example, you may want to restart a test from the beginning, or skip a step entirely and continue with the next step in the test.

1. Types of all **Check points** in QTP?

**Ans: Just give names of remembered Check points of the following.**

In QTP, there are total 8 types of check point. They are as follows:

1. **(SCP)Standard check point:It checks the GUI objects properties**

Standard Checkpoint checks the property value of an object in your application or Web page. The standard checkpoint checks a variety of objects such as buttons, radio buttons, combo boxes, lists, etc. For example, you can check that a radio button is activated after it is selected or you can check the value of an edit field.  
Standard checkpoints are supported for all add-in environments

1. **(BCP)Bitmap check point: it checks the bitmaps in an application. It takes perticular area as a bitmap.But Image check point checks only web application**

Bitmap Checkpoint checks an area of your Web page or application as a bitmap. For example, suppose you have a Web site that can display a map of a city the user specifies. The map has control keys for zooming. You can record the new map that is displayed after one click on the control key that zooms in the map. Using the bitmap checkpoint, you can check that the map zooms in correctly. Bitmap checkpoints are supported for all add-in environments.

1. **(TACP)Text area check point:It checks the perticular text area in a application**  
   **Text Area Checkpoint** checks that a text string is displayed within a defined area in a Windows application, according to specified criteria.  
   Text area checkpoints are supported for Standard Windows, Visual Basic,  
   and ActiveX add-in environments
2. **(TCP)Text check point:It checks the text in a application**  
   Text Checkpoint checks that a text string is displayed in the appropriate place in your application or on a Web page.  
   Text checkpoints are supported for all add-in environments
3. **(ACP)Accessibility Check point: It can checks the accessibility of the web application only.**   
   **Accessibility Checkpoint** identifies areas of your Web site that may not conform to the World Wide Web Consortium (W3C) Web Content  
   Accessibility Guidelines.  
   Accessibility checkpoints are supported for the Web environment
4. **Page Checkpoint** checks the characteristics of a Web page. For example, you can check how long a Web page takes to load or whether a Web page contains broken links.  
   Page checkpoints are supported for the Web environment
5. **(DCP)Database Check point:It checks the backend data of the application**  
   Database Checkpoint checks the contents of a database accessed by your Web site.
6. **(XCPF)XML check point from file: It checks the XML code for the perticular file.**  
   XML Checkpoint checks the data content of XML documents in XML files or XML documents in Web pages and frames. For more information on XML checkpoints.  
   XML checkpoints (Web page/frame) are supported for the Web  
   environment; XML checkpoints (file) are supported by all environments.
7. **Descriptive Programming** and it’s Syntax

**Ans:**

Descriptive programming can be done in two ways:

Static: We provide the set of properties and values that describe the object, directly.

Dynamic: We have to add a collection of properties and values to a description object and then provide the statement with the description object's name.

Static is easier but Dynamic provides more power, efficiency, and flexibility.

**Syntx:**

Class (“Object unique property name**:=**Object property unique value”)

1. Different between VB Script **Function procedure and Sub procedure**

**Ans:** The following are the differences between Function procedure and Sub procedure.

1. The main difference is function returns a value where as a Sub-procedure does not.
2. Sub procedure is faster in Executing/Compiling the code.
3. VB Script Inbuilt Functions which refer the Dates and Times.

**Ans:**

|  |  |
| --- | --- |
| Date Function | Returns the current system date. |
| DateAdd Function | Returns a date to which a specified time interval has been added. |
| DateDiff Function | Returns the number of intervals between two dates. |
| DatePart Function | Returns the specified part of a given date. |
| DateSerial Function | Returns a **Variant** of subtype **Date** for a specified year, month, and day. |
| DateValue Function | Returns a **Variant** of subtype **Date**. |
| Day Function | Returns a whole number between 1 and 31, inclusive, representing the day of the month. |
| Eval Function | Evaluates an expression and returns the result. |
| FormatDateTime Function | Returns an expression formatted as a date or time. |
| CDate Function | Returns an expression that has been converted to a Variant of subtype Date. |

1. What is the use of **ByVal** and **ByRef** arguments

**Ans:**

**Passing Simple Variables ByRef And ByVal**

Passing ByRef or ByVal indicates whether the actual value of an argument is passed to the CalledProcedure by the CallingProcedure, or whether a reference (called a pointer in some other languages) is passed to to the CalledProcedure. If an argument is passed ByRef, the memory address of the argument is passed to the CalledProcedure and any modification to that parameter by the CalledProcedure is made to the value in the CallingProcedure. If an argument is passed ByVal, the actual value, not a reference to the variable, is passed to the CalledProcedure.

A simple example will illustrate this clearly:

**Sub CallingProcedure()**

**Dim A As Long**

**Dim B As Long**

**A = 123**

**B = 456**

**Debug.Print "BEFORE CALL = A: " & CStr(A), "B: " & CStr(B)**

**CalledProcedure X:=A, Y:=B**

**Debug.Print "AFTER CALL = A: " & CStr(A), "B: " & CStr(B)**

**End Sub**

**Sub CalledProcedure(ByRef X As Long, ByVal Y As Long)**

**X = 321**

**Y = 654**

**End Sub**

In the CallingProcedure the variables A and B are assigned the values 123 and 456, respectively. These values are confirmed with the first Debug.Print statement. Then, the CalledProcedure is called passing arguments A and B. Within CalledProcedure the parameters X and Y as assigned the values 321 and 654 respectively, and control is returned back to the procedure CallingProcedure. Since the parameter X was declared with ByRef, a reference or pointer to A was passed to CalledProcedure and any modification to the X parameter in CalledProcedure affects the variable A in CallingProcedure. The parameter Y was declared with ByVal, so only the actual value of B was passed to CalledProcedure. Changes made to the parameter Y are not made to the varialbe B. This is illustrated by the second Debug.Print statement. This shows that A as modified by CalledProcedure but that B was not changed.

1. What is the difference between Regression testing and Re-testing

**Ans:** Regression is a type of repetitive testing, which is performed on the areas (suspected to be

prone to bugs due to the Bug fixes/change in the requirements.) **which are affected**, due to the Bug fixes/change in the requirements.

Whereas the Re-testing is a type of repetitive testing, which is performed on the areas

**which are having** the Bug fixes/change in the requirements.

1. What is Priority and Severity?

**Ans:**

|  |  |
| --- | --- |
| **Severity** | **Priority** |
| Impact of the defect on the Application | Impact of the defect w.r.t users |
| Assigned by Testers. |  |
| Severity is seriousness of the bug | Priority is How fast the bug should be rectified |
| severity describes seriousness of the defect with respect  to functionality | priority describes importance of defect to solve with respect to customer. |
| impact of the bug on the functionally | order of important of the bug |
| severity is completely technical impact driven. | priority is completely business impact driven |

Bug classification based on severity and priority

Bugs are inevitable in any software development life cycle. Everybody knows that. However, you have a deadline to meet. You have a release to get over with. There comes a point in time for a release when you take a call and decide what bugs to fix and what not to fix.

Here comes the importance of bug classification. The decision on what bugs to fix and what not to fix is made based on the bug classification.

Bug classification is done primarily to separate the most important bugs from the not so important ones.

Bug severity and bug priority are two terms which are confusing to many people. For the benefit of testers who have any doubts in this regard, here's how me and my company use these terms to properly classify a bug.

Bug severity describes how much damage a bug is doing.

Bug priority describes how important it is to fix that bug.

At first glance these two might appear the same. But they are different. They give a good picture of what to do when used together.

Consider these scenarios:

1. A bug may do a good deal of damage. But if it rarely occurs, it may be less important to fix
2. A bug may do some damage. But if it occurs consistently, it may be more important to fix

Let's see in more detail to understand how bug severity is different from bug priority ( taken the example of options available on bugzilla )

**How to determine bug severity**

|  |  |
| --- | --- |
| **Severity** | **How much damage the bug does** |
| Blocker | Application or major section freezes, crashes, or fails to start. Data is corrupted. |
| Critical | Key feature does not work, cannot be used, or returns incorrect results. |
| Major | Key feature is difficult to use or looks terrible. A secondary feature does not work, cannot be used, or returns incorrect results |
| Normal | Secondary feature is difficult to use or looks terrible. Minor feature does not work, cannot be used, or returns incorrect results |
| Minor | Secondary feature has a cosmetic issue. Minor feature is difficult to use or looks bad. |
| Trivial | Minor glitches in images, not so obvious spell mistakes, etc |
| Enhancement | Improvement to product features due to bad usability or based on feedback from users. This is not part of product requirements / design originally. |

 (A key feature is one of the main tasks a product is supposed to accomplish. A secondary feature is one that the product is expected to have, one that helps accomplish the main tasks. A minor feature is a cosmetic feature or one that gives the user some minor piece of information.)

How to set bug priority

|  |  |  |
| --- | --- | --- |
| **Priority** | **Priority description** | **Which severity bugs** |
| P1 | Must be fixed | All blockers and criticals. Majors affecting many customers. |
| P2 | Highly desirable, scheduled, but not stop ship | Majors affecting some customers. Normals affecting many customers. |
| P3 | Nice to have, but not scheduled | Normals affecting some customers. Minors and trivials that affect many customers |
| P4 | Not likely to be fixed soon | Minors and trivials affecting a few customers |

1. How much comfortable in VB scripting?

**Ans**: 8.5/10

1. What is joins? Different types of joins? How do you use types of joins like Outer join, inner joins, and Self joins?

**Ans**: The JOIN keyword is used in an SQL statement to query data from two or more tables, based on a relationship between certain columns in these tables.

The types of JOINS.

* **JOIN**: Return rows when there is at least one match in both tables
* **LEFT JOIN**: Return all rows from the left table, even if there are no matches in the right table
* **RIGHT JOIN**: Return all rows from the right table, even if there are no matches in the left table
* **FULL JOIN**: Return rows when there is a match in one of the tables

You can use a self-join to simplify nested SQL queries where the inner and outer [queries](http://databases.about.com/library/glossary/bldef-query.htm) reference the same [table](http://databases.about.com/library/glossary/bldef-table.htm). These [joins](http://databases.about.com/cs/administration/g/join.htm) allow you to retrieve related records from the same table. The most common case where you'd use a self-join is when you have a table that references itself.

1. What are Primary Key and Foreign Keys? What is the difference between Primary Key and Foreign Key?

**Ans**:

**Primary Key:**

A **primary key** is a field or combination of fields that uniquely identify a record in a table, so that an individual record can be located without confusion.

**Foreign Key:**

A **foreign key** (sometimes called a referencing key) is a key used to link two tables together. Typically you take the primary key field from one table and insert it into the other table where it becomes a foreign key (it remains a primary key in the original table).

More complicated but fuller explanation:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Employee Table | | | | |
| EmployeeID (PK) | FirstName | LastName | Department | Manager |
| 001 | Stan | Smithers | IT Support | Stan Smithers |
| 002 | Joe | Bloggs | Sales | Joe Bloggs |
| 003 | Mark | Richards | Sales | Joe Bloggs |
| 004 | Jenny | Lane | Marketing | Jenny Lane |
| 005 | Sally | Holmes | Sales | Joe Bloggs |
| 006 | John | Lee | IT Support | Stan Smithers |

A primary key is the field(s) (a primary key can be made up of more than one field) that uniquely identifies each record, i.e. the primary key is unique for each record and the value is never duplicated in the same table, so in the above table the EmployeeID field would be used. A **constraint** is a rule that defines what data is valid for a given field. So a **primary key constraint** is a rule that says that the **primary key** fields cannot be null and cannot contain duplicate data.

The problem with the above table is that you have repeating information in the manager field, this causes all sorts of problems, e.g. Fred Bloggs leaves and Jenny Smith becomes sales manager, you now have to replace all entries that say Fred Bloggs with Jenny Smith.

If however you split the last two fields out to make a department table you would only need one entry for each department, when a manager changes you only need to make the change in one place, if you setup a primary key of DeptID in the department table you would have the following.

|  |  |  |
| --- | --- | --- |
| Department Table | | |
| DeptID (PK) | Department | Manager |
| 01 | IT Support | Stan Smithers |
| 02 | Sales | Joe Bloggs |
| 03 | Marketing | Jenny Lane |

|  |  |  |
| --- | --- | --- |
| Employee Table | | |
| EmployeeID (PK) | FirstName | LastName |
| 001 | Stan | Smithers |
| 002 | Joe | Bloggs |
| 003 | Mark | Richards |
| 004 | Jenny | Lane |
| 005 | Sally | Holmes |
| 006 | John | Lee |

You now need to link the two table together so you know which department each employee is in, so what you do is take the primary key from the department table and insert it into the employee table (where it becomes a foreign key as a foreign key is the primary key from one table inserted into another table to link them).

|  |  |  |  |
| --- | --- | --- | --- |
| Employee Table | | | |
| EmployeeID (PK) | FirstName | LastName | DeptID (FK) |
| 001 | Stan | Smithers | 01 |
| 002 | Joe | Bloggs | 02 |
| 003 | Mark | Richards | 02 |
| 004 | Jenny | Lane | 03 |
| 005 | Sally | Holmes | 02 |
| 006 | John | Lee | 01 |

A **foreign key constraint** specifies that the data in a **foreign key** must match the data in the primary key of the linked table, in the above example we couldn't set the DeptID in the Employee table to 04 as there is no DeptID of 04 in the Department table. This system is called **referential integrity**, it is to ensure that the data entered is correct and not orphaned (i.e. there are no broken links between data in the tables)

The other added advantage is that you are saving space, if the following were the field sizes for the tables we have:

* EmployeeID = 3 characters
* Firstname = 10 characters
* Surname = 10 characters
* Department = 10 characters
* DeptID = 2 characters
* Manager = 20 characters

The original Employee Table would take 53 characters per record, 6 records gives us 318 characters.

The latest version of the Employee Table would take 25 characters, 6 records gives us 150 characters. The Department table would take 32 characters and there a 3 records so 96 characters, so 150+96 = 246 characters.

So over a very simple structure with just 6 records we have saved ourselves 72 characters, which would be 72 Bytes.

Doesn't sound much on 6 records but if we had 600 employees the original system would take 53\*600 = 31800 characters. Whereas the new system would take 25\*600 = 15000 + 32\*3 = 96

Which is a total of 15096 characters, a saving of 16704 characters so **we have saved over 50% of the storage space**.

1. Describe about STLC??

**Ans**:

Software Test Life Cyle:

1. Test Initiation

2. Test Planning

3. Test Design

4. Test Execution.

5. Test Report

6. Test Execution Closure.

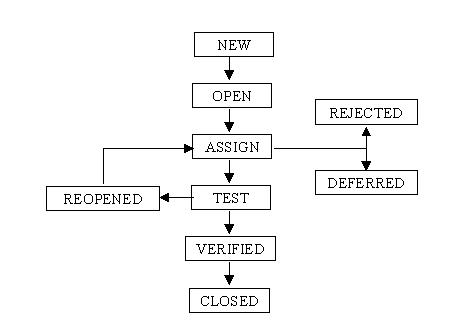
**Refer the link**: <http://www.coolinterview.com/interview/15335/>

1. Describe about Bug/Defect life cycle??

**Ans**: Describe about the Life cycle of the Bug in your project. The following info can help you in describing about the Life cycle of the Bug in your project

#### Bug Life Cycle:

In software development process, the bug has a life cycle. The bug should go through the life cycle to be closed. A specific life cycle ensures that the process is standardized. The bug attains different states in the life cycle. The life cycle of the bug can be shown diagrammatically as follows:



The different states of a bug can be summarized as follows:

1. New   
2. Open   
3. Assign   
4. Test   
5. Verified   
6. Deferred   
7. Reopened   
8. Duplicate   
9. Rejected and   
10. Closed

#### Description of Various Stages:

**1. New:** When the bug is posted for the first time, its state will be “NEW”. This means that the bug is not yet approved.

**2. Open:** After a tester has posted a bug, the lead of the tester approves that the bug is genuine and he changes the state as “OPEN”.

**3. Assign:** Once the lead changes the state as “OPEN”, he assigns the bug to corresponding developer or developer team. The state of the bug now is changed to “ASSIGN”.

**4. Test:** Once the developer fixes the bug, he has to assign the bug to the testing team for next round of testing. Before he releases the software with bug fixed, he changes the state of bug to “TEST”. It specifies that the bug has been fixed and is released to testing team.

**5. Deferred:** The bug, changed to deferred state means the bug is expected to be fixed in next releases. The reasons for changing the bug to this state have many factors. Some of them are priority of the bug may be low, lack of time for the release or the bug may not have major effect on the software.

**6. Rejected:** If the developer feels that the bug is not genuine, he rejects the bug. Then the state of the bug is changed to “REJECTED”.

**7. Duplicate:** If the bug is repeated twice or the two bugs mention the same concept of the bug, then one bug status is changed to “DUPLICATE”.

**8. Verified:** Once the bug is fixed and the status is changed to “TEST”, the tester tests the bug. If the bug is not present in the software, he approves that the bug is fixed and changes the status to “VERIFIED”.

**9. Reopened:** If the bug still exists even after the bug is fixed by the developer, the tester changes the status to “REOPENED”. The bug traverses the life cycle once again.

**10. Closed:** Once the bug is fixed, it is tested by the tester. If the tester feels that the bug no longer exists in the software, he changes the status of the bug to “CLOSED”. This state means that the bug is fixed, tested and approved.

While defect prevention is much more effective and efficient in reducing the number of defects, most organization conducts defect discovery and removal. Discovering and removing defects is an expensive and inefficient process. It is much more efficient for an organization to conduct activities that prevent defects.

#### Guidelines on deciding the Severity of Bug:

Indicate the **impact** each defect has on **testing efforts** or users and administrators of the application under test. This information is used by developers and management as the **basis** for assigning **priority** of work on defects.

A sample guideline for assignment of Priority Levels during the product test phase includes:

1. **Critical / Show Stopper** — An item that prevents further testing of the product or function under test can be classified as Critical Bug. No workaround is possible for such bugs. Examples of this include a missing menu option or security permission required to access a function under test.   
   .
2. **Major / High** — A defect that does not function as expected/designed or cause other functionality to fail to meet requirements can be classified as Major Bug. The workaround can be provided for such bugs. Examples of this include inaccurate calculations; the wrong field being updated, etc.   
   .
3. **Average / Medium** — The defects which do not conform to standards and conventions can be classified as Medium Bugs. Easy workarounds exists to achieve functionality objectives. Examples include matching visual and text links which lead to different end points.   
   .
4. **Minor / Low** — Cosmetic defects which does not affect the functionality of the system can be classified as Minor Bugs.
5. What is the Automation framework you are using??

**Ans**: We are using Some XXXXXX\* frame work.

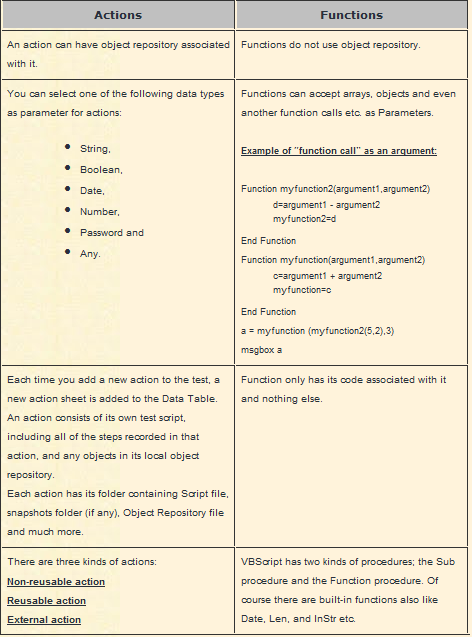
XXXXXX\* = Test script modularity frame work or Keyword driven frame work or Hybrid frame or HCL internal Automation frame work based on your comfortableness.

1. What is XXXXXX\* frame work???

**Ans**: Describe about its folder structure and architecture and explain how it is making you to perform Automation testing efficiently and effectively.

1. What is the difference between Function and Action in QTP???

**Ans**:



1. What is object repository? How many types are there? What is Shared OR? What is Local OR? What is the difference between Shared and Local ORs? How do you convert the Local OR to Shared OR?

**Ans:**

An **object** as we know is a graphic user element in an application e.g. a button or a list or an edit box and the special characteristics of an object within the QuickTest are called **object properties**. QTP stores the recorded object properties in **Object Repository**.

In this article I will be talking about **Object Repositories** to an extent.

[For more information on **Object Repositories** you can see the QTP user guide.]

**Object Repositories** are of two types **Local and shared** .

If objects are stored in a **Local Object Repository** then these are available to specific actions but not to all the actions. But if these objects are stored in one or more **Shared Object Repositories** then multiple actions or tests can use them.

By default QTP makes and uses **Local Object Repository**. If we create a new blank test and do a recording on it, QTP automatically creates a **Local Object Repository** for that test or action and stores the information about any object it finds in that corresponding Object Repository.  
  
In QTP 9 we can associate multiple **Shared Object Repositories** with an action. If multiple **Shared Object Repositories** are associated with an action then also while recording QTP stores objects in corresponding **Local Object Repository** on the condition that those objects are not already stored in any corresponding associated **Shared Object Repositories**. This is the default that every time we create a new action QTP creates a new corresponding **Local Object Repository**. It is also true that Object Repositories are associated with actions and no matter how many times we learn or record on the same object in our application in different actions the object will be stored as separate test object in each of the Local Object Repository. Local Object Repository is automatically saved with the test when we save it. The extension of the **Local Object Repository** is .mtr, but it is not accessible as a separate file as in case of the **Shared Object Repository**.

We can also manipulate some aspects of **Local Object Repository** using Quick test Object Repository Automation Object Model.  
For example we can add, remove, rename test objects in Local Object Repository. [QuickTest Object Repository Automation documents the Object Repository automation object model that enables you to manipulate QuickTest object repositories and their contents from outside of QuickTest.]

When we open a test that was created using a version of QTP earlier that version 9 we are asked whether we want to convert it or view it in read only format. In any case if the test previously used per-action Object Repository, the objects in each per action repository are moved to the Local Object Repository of each action in the test.

If the test previously used a **shared object repository**, the same shared object repository is associated with each of the actions in the test, and the **local object repository** is empty.

While learning or recording we can specify **Shared Object Repository** for the selected action. We can specify and associate one or more Shared Object Repositories with each action. We can also create new Shared Object Repository and associate it with our action.

In case of Shared Object Repository, QTP uses existing information and does not add objects to the Object Repository if we record operations on an object that already exists either in Shared or Local Object Repository.

As said earlier QTP does not add objects directly to the associated Shared Object Repository as we record, instead it adds new objects in Local Object Repository (if that object does not already exist in associated Shared Object Repository).

We can surely export Local objects to Shared Object Repository.

There are different ways in which we can move objects from **Local Object Repository** to **Shared Object Repository**:

1)  
Exporting the objects to the Shared Object Repository from the Local Object Repository:

In Object Repository window choose the action whose local objects you want to move. Choose File-> Export Local Objects.Select the location in which you want to save the file. Click on save.

2)  
We can update the **Shared Object Repository** with the **Local Object Repository:**

If we create a new test it will be created with **Local Object Repository**, we can associate any new or old Shared Object Repository with it, and so we can update that Shared Object Repository with Local Object Repository.

In Object Repository Manager open the Shared Object Repository (clear open in read only check box).  
  
The test in this case should not be open. In Object Repository Manager go to Tools –> Update From Local Repository. Select the test who's Local Object Repository you want to use. Click update all. It will move all the objects to the Shared Object Repository.

3)  
We can also merge objects from two Object Repositories (called as primary and secondary in QTP 9) into a new single Object Repository (target Object Repository in QTP 9). The original source files are not changed. It also enables you to merge objects from Local Object Repository of one or more action(s) into a Shared Object Repository. It is recommended to use as a primary Object Repository the file in which you have invested alot of your effort, like which has more number of objects.

If we do not specify a file extension for Shared Object Repository when creating a new Shared Object Repository QTP automatically appends a default extension name for Shared Object Repository as .tsr. This means that we can create Shared Object Repository with any extension other than .tsr, it should work fine (I have tried that and it works fine), I think it may create problems while merging two Object Repositories (I haven't tried that yet).

We can compare two Object Repositories using the **Object Repository Comparison Tool**. The tool enables you to identify similarities, variations or changes between two Object Repositories.

We can also copy objects to **Local Object Repository** from the **Shared Object Repository**. We can copy, paste and move objects in **Local Object Repository** and copy, paste and move objects within **Shared Object Repository** and between **Shared Object Repositories**.As said earlier we can also copy objects from **shared Object Repository** to Local Object **Repository** to modify them locally.  
We cannot remove an association between the action and its **Local Object Repository**.

The answer to the question which I asked above is:

According to QTP user guide:  
You can associate as many object repositories as needed with an action, and the same object repository can be associated with different actions as needed. You can also set the default object repositories to be associated with all new actions in all tests.

Whenever we make any changes to an Object Repository those changes are automatically updated in all the associated tests open on the same computer as soon as we make the change even if the **Object Repository** is not yet saved and if we close the same **Object Repository** without saving the changes the changes are rolled back in any open tests. For the test that was not open when we changed Object Repository, when we open the test on the same machine on which we modified the Object Repository the test is automatically updated with all the saved changes. To see saved changes in a test or repository open on a different computer, you must open the test or **object repository** file or lock it for editing on your computer to load the changes.

**Important points about Object Repositories**

It is a point to consider while planning and creating test that how you want to store objects; either you want to store them in **Local Object Repository** or **Shared Object Repository**.  
1)  
For each action, we can also use a combination of objects from the Local and Shared Object Repositories, according to our needs. Local objects can also be transferred to a shared object repository, if necessary. This will cut maintenance and increase the reusability of the tests because it will enable us to maintain the objects in a single, shared location instead of multiple locations.

2)  
If there is a same name object in both the Local Object Repository and in a **Shared Object Repository** associated with the same action, the action uses the local object definition i.e. the local object is given preference over the shared object. If an object with the same name is stored in more than one Shared Object Repository associated with the same action, the object definition is used from the first occurrence of the object, according to the order in which the **Shared Object Repositories** are associated with the action.

3)  
When we open an existing test, it always uses the object repositories that are specified in the Associated Repositories tab of the Action Properties dialog box or in the Associate Repositories dialog box. When we access **Shared Object Repositories** from tests they are read-only; we can edit them only using the Object Repository Manager.

4)  
As and when an application under test changes, such as when a "Log in" button is renamed "Sign Into," we can make one update to an XML-based **Shared Object Repository** (within the new Object Repository Manager), and the update will circulate (propagate) to all tests that reference this object. QuickTest Professional keeps object-level changes synchronized among users throughout test creation efforts.

**Object Repository dialog box**  
Object Repository dialog box window shows a tree of all the objects (either Local or Shared) on its left hand side in the selected action. On selecting any object in the tree Object Repository window shows the information about the object like the name, repository in which it is stored etc. On the left hand side in a tree local objects are editable while the shared ones are grayed out (non-editable).

To **view** the test object properties, to **modify** test object properties and to **add** objects to Local Object Repository we can use Object Repository window. We can also **delete** objects from Object Repository window; this is needed as when an object is removed form the test it is not automatically removed from the **Local Object Repository**.

**Object Repository** in QTP is XML based means that if we change something related to the object in **Shared Object Repository.**, the change will be propagated to all the tests that reference this object, in real time.

**Adding Objects to Repositories**  
[Please see QTP user guide for in-depth information on these below points.]  
We can add objects to **Shared Object Repository** or **Local Object Repository** in a number of different ways-  
We can decide whether to add only a selected object, or to add all objects of a certain type, such as all button objects, or to add all objects of a specific class, such as all WebButton objects.

We can modify objects stored in a **Local Object Repository** using the Object Repository Window and objects in a **Shared Object Repository** using the Object Repository Manager.  
It is possible to add objects to the object repository before they exist in an application.  
We can also add objects to the **Local Object Repository** while *editing our test*.  
We can add the object directly to a **Shared Object Repository** using the Object Repository Manager, so that it is available in all actions that use this **Shared Object Repository.**  
If needed, we can merge test objects from the **Local Object Repository** into a **Shared Object Repository.**

We can also add objects to a **Shared Object Repository** while navigating through the application ("Adding Objects Using the Navigate and Learn Option").  
We can also add an object to the **Local Object Repository** by choosing it from the application in the Select Object for Step dialog box (from a new step in the Keyword View or from the Step Generator).

We can add objects to the **object repository** using the Add Objects to Local or Add Objects option.  
We can add objects to the **Local Object Repository** of the current action by selecting the required object in the Active Screen.

**Object Repository:** displays a tree of all objects in the current component or in the current action or entire test( depending on the object repository mode you selected).   
we can view or modify the test object description of any test object in the repository or to add new objects to the repository.   
Quicktest learns the default property values and determines in which test object class it fits.If it is not enough it adds assistive properties, one by one to the description until it has compiled the unique description. If no assistive properties are available, then it adds a special Ordinal identifier such as objects location on the page or in the source code.

When you create test all the information about the objects in your test is stored in object repository. There are two types of object Repositories   
1. Shared object repository   
2. Per action repository

Test objects can be stored in two types of object repositories—a **shared object repository** and a **local object repository**. A shared object repository stores test objects in a file that can be accessed by multiple components (via their application areas) in read-only mode. A local object repository stores objects in a file that is associated with one specific component, so that only that component can access the stored objects.

 When you plan and create components, you must consider how you want to store the objects in your components. You can store the objects for each component in its corresponding local object repository, or you can store the objects in your components in one or more shared object repositories. By storing objects in shared object repositories and associating these repositories with your components’ application areas, you enable multiple components to use the objects. For each component, you can use a combination of objects from your local and shared object repositories, according to your needs. You can also transfer local objects to a shared object repository, if required. This reduces maintenance and enhances the reusability of your components because it enables you to maintain the objects in a single, shared location instead of multiple locations.

 If you are new to using QTP, you may want to use local object repositories. In this way, you can record and run components without creating, choosing, or modifying shared object repositories because all objects are automatically saved in a local object repository that can be accessed by its corresponding component. If you modify an object in the local object repository, your changes do not have any effect on any other component.

 If you are familiar with testing, it is probably most efficient to save objects in a shared object repository. In this way, you can use the same shared object repository for multiple components—if the components include the same objects. Test object information that applies to many components is kept in one central location. When the objects in your application change, you can update them in one location for all the components that use this shared object repository.

 If an object with the same name and description is located in both the local object repository and in a shared object repository associated with the same component, the component uses the local object definition. If an object with the same name and description is located in more than one shared object repository associated with the same component, the object definition is used from the first occurrence of the object, according to the order in which the shared object repositories are associated with the component.

 Local objects are saved locally with the component, and can be accessed only from that component. When using a shared object repository, you can use the same object repository for multiple components. You can also use multiple object repositories for each component.

 When you open and work with an existing component, it always uses the object repositories that are specified in the application area with which the component is associated. Shared object repositories are read-only when accessed from components; you edit them using the Object Repository Manager.

1. What is a User defined Function and when will you use it and give an example for User defined functions?

**Ans:** A **User-Defined Function**, or **UDF**, is a function provided by the user of a program or environment, in a context where the usual assumption is that functions are built into the program or environment.

Using QuickTest you can define and store your user-defined functions either in a function library (saved as a **.qfl** file by default) or directly in an action within a test. A function library is a Visual Basic script containing VBscript functions subroutines modules and so forth. You can also use QuickTest to modify and debug any existing function libraries (such as **.vbs** or **.txt** files).

When you store a function in a function library and associate the function library with a test the test can call the public functions in that function library. Functions that are stored in an associated function library can be accessed from the Step Generator (for tests and function libraries) and the **Operation** column in the Keyword View as well as being entered manually in the Expert View.

When you store a function in a test action it can be called only from within that action—the function cannot be called from any other action or test. This is useful if you do not want the function to be available outside of a specific action.

You can also define private functions and store them in a function library. Private functions are functions that can be called only by other functions within the same function library. This is useful if you to reuse segments of code in your public functions

User defined function is nothing but we write our own functions to perform set of action which contains a login to execute some steps on the application without making use of inbuilt qtp functions or which contain full logical code by us.

For. Eg:

If you need to do some arithmetical operation and just return the value to some object (textbox) or something else. Then write a function on ur own and pass the value to do the operation and that's it.

Functions - two types

1)Buitin - Prepared by QTP

2)User Defined - Prepared by programmer/tester

Syntan to creat User defined function in QTP

Function FunctionName(Parameter1, Param2,.....)

Statement1

Statement2

.....

.....

End Function

Example:

Function Bigger(value1, value2)

If value1 > value2 Then

Bigger = value1

Else

Bigger = value2

End If

End Function

Calling the function in the script

Result = Bigger(20,21)

Msgbox Result 'gives 21

User defined function is a peice of code/program which intended to perform some specific task.

e.g if i want to create 100 user id's and i can use each id to create a profile. To avaoid duplication of ID's we can write a function (.vbs extension) which generate ID's based on system date and time so all the ID's will unique. After that we need to add that function into resources tab in QTP and can call that function using a single line statement as ...call <functionname>(arguments, if any)