1. The UVM framework has phases to synchronize the major functional steps in a test run. These sequential steps are broadly divided into build-phases, run-time phases and clean-up phases. Objections in UVM are used to ensure that the simulation doesn't stop prematurely. Since the run-time phases are the only phases that take "time", objections are used only with those phases. A UVM component or sequence will raise a phase objection at the beginning of an activity that must be completed before the phase stops, so the objection will be dropped at the end of that activity. Once all of the raised objections are dropped, the phase terminates.

2. A virtual interface is a pointer to an actual interface in SystemVerilog. It allows for abstraction of the test program from the actual signals in the design. In a UVM testbench, virtual interfaces are useful inside the class definitions (eg. driver and monitor) to access the actual interface connected to the DUT for the purposes of reading/driving the signals.

3. All UVM phases apart from the run-phase execute in zero time. All the time consuming activities (driving the test sequences onto the DUT, monitoring the output from the DUT) happen in the run phase. Therefore, the run phase should be defined as a task. All the other phases should be defined as functions.

4. The operations in a phase are carried out in top-down or bottom-up orders. In a top-down phase, the component tree is traversed in a top-down fashion, while calling the execute command of each component. On the other hand, in a bottom-up phase, the traversal is depth-first and the child components execute first.

   The build phase is executed in Top-Down fashion since the parent object instantiates the child object in its own build phase. This also allows the parent objects to provide override settings which the children use when executing their own build phase.

5. **Sequencer** : Creates the transactions, randomizes them and sends them to the driver. The sequencer is the primary stimulus generator.
   Driver : Receives the transaction from the sequencer and then drives the signals to the DUT (possibly through a virtual interface).
   Monitor : The monitor is a passive component that also has access to the DUT interface signals. It can provide protocol checks on the signals and send them as a transaction to the subscribers and scoreboards in the design for coverage measurement and correctness check.
   Yes, the driver can perform the functions on the Monitor. However, this will not be a good idea as it harms the scalability of the different components. The monitor is a passive component, and it can ideally be used as a passive checker in cases when the DUT is not being driven by the same driver (Eg. the DUT being used inside a bigger module, and therefore, being driven internally by the bigger design). Using the driver for these functions harms the reusability of the testbench components.

6. An example testbench for testing such a module has been given in Lab3. The same testbench is capable of testing the logical unit.

   For a pipelined design, FIFO's can be added to the ports of the scoreboard to handle the different latency of the inputs and outputs. These have been added in the testbench given for Lab3. The monitor will also start sampling late to account for the initial delay in filling the pipe.