**Session 1 :**

**What is UVM ?**

1. UVM stands for *U*niversal *V*erification *M*ethodology. It is a standardized methodology for verifying digital designs and systems-on-chip (SoCs) in the semiconductor industry
2. UVM is built on top of the SystemVerilog language and provides a framework for creating modular, reusable testbench components that can be easily integrated into the design verification process.
3. It also includes a set of guidelines and best practices for developing testbenches, as well as a methodology for running simulations and analyzing results.
4. UVM has become the de facto standard for design verification in the semiconductor industry, and is widely used by chip designers and verification engineers to verify the correctness and functionality of their designs.

**What was use before UVM ?**

1. **OVM :** (Open Verification Methodology) was introduced in 2008 as an open-source verification methodology for digital designs and systems-on-chip (SoCs) and was based on SystemVerilog
2. **UVM :** was introduced in 2011 as a successor to OVM, and it built upon the concepts and principles of OVM. UVM was designed to be a more standardized and flexible methodology that could be easily adapted to different verification environments and use cases

**Why was OVM replaced by UVM ?**

1. **Standardization:** OVM was an open-source methodology that lacked a formal standard, which made it difficult for different organizations and tools to interoperate effectively. UVM was designed to be a more standardized methodology, with a well-defined standard that could be adopted by the entire semiconductor industry.
2. **Flexibility:** OVM was designed primarily for transaction-level modeling (TLM), which limited its applicability to other verification scenarios, such as register-transfer level (RTL) modeling. UVM was designed to be more flexible, with support for both TLM and RTL modeling, as well as other verification scenarios.
3. **Reusability:** OVM provided a set of pre-defined classes and components for creating testbenches, but these components were not always modular and reusable. UVM was designed to be more modular and reusable, with a clear separation between testbench components and the design-under-test (DUT).
4. **Maintainability :**  OVM was not always easy to maintain or update, as changes to the methodology could affect existing testbenches and components

UVM was designed to be more maintainable, with a clear separation between the methodology and the implementation of testbenches and components.

**What does UVM contains ?**

1. **Testbench Components:** UVM provides a set of base classes that can be extended to create testbench components, such as drivers, monitors, scoreboards, and agents.
2. **Transactions:** Transactions are used to model the communication between the design-under-test (DUT) and the testbench. UVM provides a transaction class that can be extended to create transaction objects that carry information between the DUT and the testbench.
3. **Phases:** UVM defines a set of simulation phases that enable users to control the order in which testbench components are created, initialized, and executed.
4. **Messaging and Reporting:** UVM provides a messaging and reporting infrastructure that enables users to output information about the simulation, such as warnings, errors, and debug information.
5. **Configuration:** UVM provides a configuration database that allows users to store and retrieve configuration information for testbench components.
6. **Functional Coverage:** UVM provides a mechanism for tracking functional coverage, which is used to ensure that the design has been thoroughly tested.
7. **Register Abstraction Layer:** UVM provides a register abstraction layer (RAL) that simplifies the process of creating and accessing register maps.