SECTION 1 — UVM BASICS

1️⃣ What is UVM? Why do we use it?

UVM (Universal Verification Methodology) is a SystemVerilog class library used to build structured, reusable, and scalable testbenches.

We use UVM because:

- It provides a standard architecture
- Enables reuse (IP → SoC)
- Supports randomization & coverage
- Helps create layered, modular TBs
- Makes debugging easier with phases, factory, reporting

2️⃣ Difference between SystemVerilog TB and UVM TB?

SV TB	UVM TB
Procedural	Class-based
Manual wiring	Standard architecture
No reuse	Highly reusable
No phases	Has phases
Stimulus is ad-hoc	Stimulus via sequences

3️⃣ What are the main components of a UVM Testbench?

- Sequence item (transaction)
- Sequence
- Sequencer
- Driver
- Monitor
- Agent
- Scoreboard
- Environment
- Test

4️⃣ What is a UVM Agent?

An **agent bundles driver + sequencer + monitor** for a particular interface.

Types:

- **Active agent** → drives + monitors
- **Passive agent** → monitors only

5️⃣ What is the difference between uvm_object and uvm_component?

uvm_object	**uvm_component**
Lightweight	Heavy (has phases)
No hierarchy	Has hierarchy
No build/connect phases	Has all phases
Created by create()	Created inside build_phase

Used for:

- **object:** sequences, transactions, configs
- **component:** env, agent, scoreboard, driver, monitor

✅ SECTION 2 — UVM PHASES

6️⃣What are the main UVM phases?

Build phase → create components
Connect phase → connect TLM ports
Run phase → time-based simulation
Extract phase → gather stats
Check phase → ensure no errors
Report phase → final report

7️⃣What do you do in each phase?

- build → create components
- connect → TLM connections

- run → stimulus, driving, sampling
- check → final checks
- report → summary

8️⃣Why is objection used in run_phase?

To keep the simulation alive.

```
phase.raise_objection(this);
// stimulus
phase.drop_objection(this);
```

✅SECTION 3 — CONFIGURATION DATABASE

9️⃣Why do we use uvm_config_db?

To pass configuration items without hardcoding paths.

🔟 Difference between set() and get()?

- `set()` → write value into DB
- `get()` → read the value

1️⃣1️⃣What happens if get() fails?

You usually call:

```
`uvm_fatal("CFG", "Failed to get config")
```

Simulation stops.

✅ SECTION 4 — UVM FACTORY & OVERRIDES

1️⃣2️⃣ What is the UVM factory?

A mechanism to create objects **indirectly** using type IDs.

Allows component substitution without modifying TB code.

1️⃣3️⃣ What is type override?

Replace all instances of a type:

```
factory.set_type_override_by_type(old::get_type(), new::get_type());
```

1️⃣4️⃣ What is instance override?

Replace only a **specific** component instance:

```
factory.set_inst_override_by_type(old::get_type(), new::get_type(),
  "env.agent.drv");
```

1️⃣5️⃣ Why does UVM use macros like `uvm_component_utils`?

Registers the component with factory

Enables printing, copying, comparing, packing

✅ SECTION 5 — SEQUENCES & DRIVERS

1️⃣6️⃣ What is a uvm_sequence_item?

A transaction object describing stimulus fields.

Example:

```
class pkt extends uvm_sequence_item;
  rand bit [7:0] data;
endclass
```

1️⃣7️⃣ How does a sequence communicate with a driver?

Using **TLM ports**:

- Driver uses seq_item_port
- Sequencer uses seq_item_export

1️⃣8️⃣ Describe the sequence → driver handshake

1. Sequence creates item
2. start_item()
3. Randomize
4. finish_item()
5. Driver receives it using get_next_item()
6. Drives DUT
7. Calls item_done()

19 What is the difference between uvm_do and start_item/finish_item?

uvm_do	start_item / finish_item
Legacy macro	Modern, preferred
Less readable	Explicit control
Not recommended	Recommended

20 What is a virtual sequencer?

A sequencer that **holds handles to multiple sequencers** (e.g., APB + UART). Used in SoC & subsystem verification.

21 What is a virtual sequence?

A sequence that runs on a virtual sequencer to coordinate multiple sequences.

```
fork
  apb_seq.start(vseqr.apb_seqr);
  uart_seq.start(vseqr.uart_seqr);
join
```

22 What are layered sequences?

Sequences controlling other sequences, used for advanced scenarios.

✅ SECTION 6 — TLM (TRANSACTION LEVEL MODELING)

2️⃣3️⃣ Types of TLM ports in UVM?

- get, put, peek
- analysis_port
- analysis_export
- analysis_imp
- seq_item_port

2️⃣4️⃣ What is an analysis port?

A **broadcast port** (one-to-many)
Used by monitors to send transactions:

```
ap.write(tr);
```

2️⃣5️⃣ What is uvm_analysis_imp?

Scoreboard implements:

```
function void write(tr);
```

2️⃣6️⃣ Why is monitor passive?

It **never drives** signals, only observes.

It samples DUT pins, creates transactions, sends through analysis port.

✅ SECTION 7 — SCOREBOARD

2️⃣7️⃣ What is the role of a scoreboard?

To check DUT correctness by comparing:

- Expected transactions
- Actual transactions

2️⃣8️⃣ How do you get expected vs actual data?

Expected → from input monitor

Actual → from output monitor

2️⃣9️⃣ What is a reference model?

Golden model used to compute expected outputs.

Example: Predict FIFO behavior with a SystemVerilog queue.

3️⃣0️⃣ When do you use uvm_tlm_analysis_fifo?

When the monitor produces data at different rate than scoreboard consumes.

✅ SECTION 8 — COVERAGE

3️⃣1️⃣ What is functional coverage?

A coverage metric that checks whether all **functional scenarios** were tested.

3️⃣2️⃣ Types of coverage

- Covergroup
- Coverpoint
- Cross coverage
- Transition bins

3️⃣3️⃣ Where do we put covergroups?

- Inside monitor (typical)
- Inside subscriber

3️⃣4️⃣ What is code coverage?

Coverage of RTL code structure (statements, branches, toggles).

Difference:

- **Code coverage** → checks DUT code exercised
- **Functional coverage** → checks verification plan completion

✅ SECTION 9 — UVM REGISTER MODEL (RAL)

3️⃣5️⃣ Why use RAL?

To model DUT registers in SystemVerilog so test can do:

```
reg_model.my_reg.write(status);
```

3️⃣6️⃣ What are main components of RAL?

- uvm_reg
- uvm_reg_block
- uvm_reg_adapter
- uvm_reg_predictor

3️⃣7️⃣ What is a register adapter?

Converts RAL read/write into bus transactions (APB, AXI).

3️⃣8️⃣ What is predictor?

Updates the mirror value of registers when DUT changes them.

✅SECTION 10 — MULTIPLE AGENTS / MULTI-UVC

3️⃣9️⃣ How do you integrate multiple agents?

Example: UART + FIFO → connect both monitors to scoreboard:

```
uart_agt.mon.ap.connect(sb.uart_imp);
fifo_agt.mon.ap.connect(sb.fifo_imp);
```

4️⃣0️⃣ How do you write a reusable env?

By parameterizing agents
Using config DB
Factory overrides

✅SECTION 11 — TEST LAYER

4️⃣1️⃣ What is a base test?

Creates environment
Applies configurations
Runs default sequences

4️⃣2️⃣ What are derived tests?

Extend base test and customize behavior

4️⃣3️⃣ How do you start a sequence from test?

```
seq.start(env.agt.seqr);
```

4️⃣4️⃣ Why does test need objections?

To keep simulation alive while sequences run.

✅ SECTION 12 — VIRTUAL INTERFACES

4️⃣5️⃣ Why do we need virtual interfaces?

To pass physical interface handles to UVM components (drivers, monitors).

4️⃣6️⃣ How to pass virtual interface?

Top:

```
uvm_config_db#(virtual my_if)::set(null, "*", "vif", if_hdl);
```

Driver:

```
uvm_config_db#(virtual my_if)::get(this, "", "vif", vif);
```

✅ SECTION 13 — REPORTING & DEBUGGING

4️⃣7️⃣ What are reporting macros?

- `uvm_info`
- `uvm_warning`

- uvm_error
- uvm_fatal

4️⃣8️⃣ What is verbosity?

Controls how much log prints:

- UVM_LOW
- UVM_MEDIUM
- UVM_HIGH
- UVM_FULL

Set by:

+UVM_VERBOSITY=UVM_HIGH

✅ SECTION 14 — MISCELLANEOUS

4️⃣9️⃣ What is RAL mirror?

Software copy of DUT register values.

5️⃣0️⃣ What is factory override priority?

Instance override has higher priority than type override.