1. What are the basic components of a module? Which components are mandatory?
    - **Module Name**: Defines the module.
    - **I/O Ports**: Input, Output, and Inout signals.
    - **Port List**: Describes the inputs and outputs of the module.
    - **Internal Signals/Wires**: Declare internal signals/wires within the module.
    - **Procedural Blocks**: Always blocks, initial blocks, and functions/tasks.
    - **Instances**: Submodules that can be instantiated inside a module.
    - Only the module declaration and the endmodule keyword are mandatory.
2. Should a module that does not interact with its environment have any I/O ports?
   No, a module that doesn't interact with its environment does not require any I/O ports. Such modules could be internal helper modules.
3. How can a race condition occur in Verilog? What are the ways to avoid the race condition?
   A race condition can occur when two or more always blocks (or processes) update the same signal at the same time. The outcome may depend on the order in which these blocks are executed, leading to unpredictable behaviour. To avoid race conditions
    - Use **non-blocking assignments (<=)** for sequential logic.
    - Avoid mixing **blocking (=)** and **non-blocking (<=)** assignments in sequential blocks.
    - Ensure a clear dependency order of always blocks.
4. What is the difference between the following lines of code?
   reg1<= #10 reg2;
   reg3=#10reg4;
   reg1 <= #10 reg2;: This is a **non-blocking assignment** with a 10-time unit delay. The update of reg1 will happen after the delay, without blocking further execution.
   reg3 = #10 reg4;: This is a **blocking assignment** with a 10-time unit delay. The assignment to reg3 happens after the delay and blocks further execution until the assignment completes.
5. Consider a 2:1 mux; what will the output F be if the select(sel) is ‘X’?
   When sel = 'X', the output F will be unknown (X), because Verilog can't resolve the selection between inputs A and B when the selector is not determinable.
6. Write Verilog code for:
    a. A parallel encoder

```
module parallel_encoder(
   input [7:0] data_in,
   output reg [2:0] data_out
);
   always @(*) begin
      case (data_in)
         8'b00000001: data_out = 3'b000;
         8'b00000010: data_out = 3'b001;
         8'b00000100: data_out = 3'b010;
         8'b00001000: data_out = 3'b011;
         8'b00010000: data_out = 3'b100;
         8'b00100000: data_out = 3'b101;
         8'b01000000: data_out = 3'b110;
         8'b10000000: data_out = 3'b111;
         default: data_out = 3'bxxx;
      endcase
   end
endmodule
```

b. A priority encoder

```
module priority_encoder_casez(
   input [7:0] data_in,
   output reg [2:0] data_out
);
   always @(*) begin
      casez (data_in)
         8'b1???????: data_out = 3'b111;  // Highest priority (bit 7 is 1)
         8'b01??????: data_out = 3'b110;  // Next priority (bit 6 is 1)
         8'b001?????: data_out = 3'b101;  // Next priority (bit 5 is 1)
         8'b0001????: data_out = 3'b100;  // Next priority (bit 4 is 1)
         8'b00001???: data_out = 3'b011;  // Next priority (bit 3 is 1)
         8'b000001??: data_out = 3'b010;  // Next priority (bit 2 is 1)
         8'b0000001?: data_out = 3'b001;  // Next priority (bit 1 is 1)
         8'b00000001: data_out = 3'b000;  // Lowest priority (bit 0 is 1)
         default: data_out = 3'bxxx;      // No valid input
      endcase
   end
endmodule
```

c. Read and write into a file

```
module file_read_write;
   integer file_in, file_out;
   reg [7:0] data;

   initial begin
      // Open file for reading
      file_in = $fopen("input_file.txt", "r");
      // Open file for writing
      file_out = $fopen("output_file.txt", "w");

      // Read data from file
      while (!$feof(file_in)) begin
         $fscanf(file_in, "%b\n", data);
         $fwrite(file_out, "Read Data: %b\n", data);
      end

      // Close the files
      $fclose(file_in);
      $fclose(file_out);
   end
endmodule
```

7. What is the difference between better compiled, interpreted, event and cycle-based simulators?
   - **Compiled Simulator**: Compiles the code into machine instructions before simulation. It is faster but consumes more memory.
   - **Interpreted Simulator**: Executes the code directly, without compiling. It is slower but requires less memory.
   - **Event-based Simulator**: Focuses on events like signal changes and triggers in the design, and processes only when events occur, leading to efficient simulation of large designs.

- **Cycle-based Simulator**: Simulates designs based on clock cycles, focusing on synchronous parts of the design, offering higher performance for specific clock-based designs.

8. Can we mix blocking and non-blocking in one always block?
   It is not recommended to mix blocking (=) and non-blocking (<=) assignments within the same always block, as it can lead to race conditions and unpredictable behavior. Use non-blocking for sequential logic and blocking for combinational logic.
9. How do we avoid Latch in Verilog?
   Ensure that all outputs of combinational logic are assigned in all possible conditions within an always block. If any path is left unassigned, the synthesis tool may infer a latch to hold the previous value.
10. How can we initialize following memory array in Verilog
    reg [7:0] my_memory[0:255];
    reg [7:0] my_memory [0:255];

    ```
    initial begin
       $readmemh("memory_data.hex", my_memory);  // Hex format initialization
    end
    ```
    Or
    ```
    initial begin
       my_memory[0] = 8'hFF;
       my_memory[1] = 8'hAA;
       // Initialize other locations as needed
    end
    ```