1. Describe the following operations with an example:
    a. Arithmetic:
        * : Multiplication
        / : Division
        + : Addition
        - : Subtraction
        % : Modulus(remainder)
    b. Logical:
        !: Logical NOT
        && : Logical AND
        || : Logical OR
    c. Relational:
        > : Greater than
        < : Less than
        >= : Greater than or equal to
        <= : Less than or equal to
    d. Equality:
        == : Equality (value comparison)
        != : Not equal
        === : Strict equality (value and type comparison)
        !== : Strict not equal
    e. Bitwise:
        ~ : Bitwise NOT
        & : Bitwise AND
        | : Bitwise OR
        ^ : Bitwise XOR
        ^~ : Bitwise XNOR
    f. Reduction:
        & : AND reduction
        ~& : NAND reduction
        | : OR reduction
        ~| : NOR reduction
        ^ : XOR reduction
        ^~ : XNOR reduction
    g. Shift:
        >> : Right shift
        << : Left shift
    h. Concatenation: { }
        { } : Concatenates two values
    i. Replication: { { } }
        { { } } : Replicates a value multiple times
    j. Conditional ?:
        ?: : Conditional operator Condition? (true_output) : (false_output)
2. Explain operator precedence in Verilog.

a. **Unary operators**: `+`, `-`, `!`, `~`, `&`, `~&`, `|`, `~|`, `^`, `~^`, `^~`
b. **Multiplication, Division, Modulus**: `*`, `/`, `%`
c. **Addition, Subtraction**: `+`, `-`
d. **Shift operators**: `<<`, `>>`
e. **Relational operators**: `<`, `<=`, `>`, `>=`
f. **Equality operators**: `==`, `!=`, `===`, `!==`
g. **Bitwise operators**: `&`, `|`, `^`, `^~`, `~^`
h. **Logical AND**: `&&`
i. **Logical OR**: `||`
j. **Conditional operator**: `?:`
k. **Concatenation**: `{}`
l. **Replication**: `{{}}`
m. **Assignment operators**: `=`, `<=`

3. Design the following
   a. 3-bit ring couter.

```
module ring_counter (
    input clk,       // Clock input
    input reset,     // Reset signal
    output reg [2:0] q // 3-bit output
);
    always @(negedge clk or posedge reset) begin
        if (reset)
            q <= 3'b001; // Initialize with 001 on reset
        else
            q <= {q[1:0], q[2]}; // Shift the bit left and circulate
    end
endmodule
```

   b. Negative-edge triggered D-FF with clear.

```
module d_ff (
    input clk,       // Clock input (negative-edge triggered)
    input clear,     // Asynchronous clear
    input d,         // Data input
    output reg q     // Output
);
    always @(negedge clk or posedge clear) begin
        if (clear)
            q <= 1'b0; // Clear the flip-flop
        else
            q <= d;    // Store the value of d
    end
endmodule
```

   c. 4-bit asynchronous counter.

```
module async_counter (
    input clk,       // Clock input
    input reset,     // Reset signal
    output reg [3:0] q // 4-bit output
```

```
);
   always @(posedge clk or posedge reset) begin
      if (reset)
         q <= 4'b0000; // Reset the counter to 0
      else
         q <= q + 1;   // Increment the counter asynchronously
   end
endmodule
```

d. 4-bit comparator which compares two 4-bit no. & prints largest no.

```
module comparator (
   input [3:0] a,   // First 4-bit input
   input [3:0] b,   // Second 4-bit input
   output reg [3:0] largest // Output the largest number
);
   always @(*) begin
      if (a > b)
         largest = a;  // If a is larger, output a
      else
         largest = b;  // Otherwise, output b
   end
endmodule
```