4. (10 points) There are four Verilog code snippets in this section. For each code, first
   state whether or not there is a mistake. If there is a mistake explain how to correct it.
   Note: Assume that the behavior as described, is correct

   (a)

```
1  module mux2 ( input [1:0] i,   output s, input z);
2     assign s= (z) ? i[1]:i[0];
3  endmodule
4
5  module one (input [3:0] data, input sel1, input sel2, output z );
6     wire [1:0] temp ;
7
8     mux2 i0 (data[1:0], sel1, temp[0]);
9     mux2 i1 (data[3:2], sel1, temp[1]);
10    mux2 final (temp, sel2, z);
11 endmodule
```

> **Solution:** This code is not correct. It uses an ordered instantiation template where
> the ordering of the pins should correspond to the order they are declared. This itself
> is not wrong, but the *input* signals on the second position (sel1,sel2) connect to the
> *output* of the instance *mux*. Either the ordering of *mux2* needs to be changed, or
> the ordering of the elements the instantiation. The better option would be to use a
> *connect by name*.

   (b)

```
1  module two (input [3:0] a, input [0:3] b, output reg [3:0] z);
2   always @ ( *)
3       if (a == 0)
4         z={b[0],b[1],b[2],b[3]};
5       else
6         z=a+b;
7  endmodule
```

> **Solution:**
>
> The code is syntactically correct. There is a lot of unnecessary code: using one input
> as [0:3] and the other as [3:0] is not the smartest choice, the entire code could be
> written as *assign z=a+b*, but syntactically the code is correct.

(c)

```verilog
module three (clk, rst, a, b, c, z);
   input a,b,c,clk,rst;
   output reg z;
   reg q;

   always @ (*)
     begin
        q <= a ^ b;
        if (c) q <= ~(a^b);
     end
   always @ (negedge clk)
        if (rst) z= 1'b0;
        else     z= q;
endmodule
```

> **Solution:** The code is syntactically correct. Once again, it is a bit unconventional. One always block use blocking statements and the other non-blocking statements. Both would work, although it would probably be more efficient to write them otherwise.

(d)

```verilog
module four (input [2:0] sel, output reg [5:0] z);
 case (sel)
   0: z = 6'b00_0000;
   1: z = 6'b00_0001;
   2: z = 6'b00_0011;
   3: z = 6'b00_0111;
   4: z = 6'b00_1111;
   5: z = 6'b01_1111;
   6: z = 6'b11_1111;
   default: z= 6'b00_0000;
 endcase
endmodule
```

> **Solution:** This code is not correct. The case statement is a sequential statement that needs to be within an always statement.