1. In order to write an Verilog HDL description of any circuit you will need to write a module which is the fundamental descriptive unit in Verilog. A module is a set of text describing your circuit and is enclosed by the keywords module and endmodule.

2. As the program describes a physical circuit you will need to specify the inputs, the outputs, the behavior of the circuit and how the gates are wired. To accomplish this, you need the keywords input, output, and wire to define the inputs, outputs and the wiring between the gates, respectively.

3. There are multiple ways to model a circuit
   - gate-level modeling,
   - data flow modeling,
   - behavioral modeling,
   - or a combination of the above.

4. A simple program modeling a circuit (see Figure 2) at the gate-level, is provided below.

Program 1 Simple program in Verilog modeling a circuit at the gate-level

```
module simple_circuit(output D,E, input A,B,C);
wire w1;

and G1(w1,A,B);
not G2(E,C);
or G3(D,w1,E);
endmodule // no semi-colon
```

5. As seen above the outputs come first in the port list followed by the inputs.

6. Single line comments begin with //

![Figure 2: Simple Circuit](http://www.ee.nmt.edu/~elosery/fall_2008/ee231L/edited_from/Dr. Ahmad Almulhem)
7. Multi-line comments are enclosed by /* */

8. Verilog is case sensitive.

9. A simple program modeling a circuit using data ow, is provided below.

Program 2 Simple program in Verilog modeling a circuit using data ow

```verilog
module simple_circuit(output D, E, input A, B, C);

assign D = (A & B) | ~C;
assign E = ~C;

endmodule
```

10. You can identifiers describing multiple bits known as vectors. For example you may write Program 2 as

Program 3 Simple program in Verilog modeling a circuit using data ow using vectors

```verilog
module simple_circuit(output [1:0] Y, input [0:2] X);

assign Y[1] = (X[0] & X[1]) | ~X[2];
assign Y[0] = ~X[2];

endmodule
```

In this example, we have the input as three bits representing A, B, C and we have denoted them as [0:2] X which means we have three bits with the index 0 representing the MSB. We could have specified it as [2:0] X in which case the index 2 represents the MSB.

11. Given an identifier [7:0] X you can assign it values by

```verilog
assign [7:0] X = 8'h00101011;
```

where the 8'b specifies that we are defining an 8-bit binary number and 00101011 is that 8-bit binary number. You can also assign parts of the number as

```verilog
assign [2:0] X = 3'b101;
```

which assigns only the last three bits of X.
Verilog - Behavioral Modeling
1. **always and reg**

   1. Behavioral modeling use the keyword always.

   2. Target output is of type `reg`. Unlike a `wire`, `reg` is updated only when a new value is assigned. In other words, it is not continuously updated as `wire` data types.

   3. `always` may be followed by an event control expression.

   4. `always` is followed by the symbol `@` followed by a list of variables. Each time there is a change in those variables, the `always` block is executed.

   5. There is no semicolon at the end of the `always` block.

   6. The list of variables are separated by logical operator `or` and not bitwise OR operator `|`.

   7. Below is an example of an `always` block.

   ```
   always @(A or B)
   .
   .
   ```

2. **if-else Statements**

   `if-else` statements provide means for a conditional output based on the arguments of the `if` statement.

3. **case Statements**

   `case` Statements provide an easy way to represent a multi-branch conditional statement.

   1. The first statement that makes a match is executed.

   2. Unspecified bit patterns could be treated using `default` keyword.
output out;
input s,A,B;
reg out;

if (s) out = A;  // if select is 1 then out is A
else out = B;    // else output is B

Program 1 four-to-one line multiplexer

module mux_4x1_example(
output reg out,
input [1:0] s,  // select is represented by 2 bits
input in_0, in_1, in_2, in_3);

always @(in_0, in_1, in_2, in_3, s)
case (s)  // no semi-colon
  2'b00:   out = in_0; // if s is 00 then output is in_0
  2'b01:   out = in_1; // else if s is 01 then out is in_1
  2'b10:   out = in_2; // else if s is 10 then out is in_2
  2'b11:   out = in_3; // else if s is 11 then out is in_3
endcase    // case statement ends with endcase
endmodule