$random :

- It is a system task
- Returns 32 bit random number
- The random number is signed integer (+ve no. and -ve no.)
- Seed parameter is used to control the number

$urandom :

- Same as $random it generated unsigned number

$urandom_range:

- Generated unsigned integer with a specific value
- EX:
    1) $urandom_range(20); //min=0 max=20
    2) $urandom_range(10,30);//min=10 max=30

HOW TO RANDOMIZED A VARIABLE

- $random and $urandom
- $urandom_range
- std::randomize
- randomize()

Program 1 : Randomized a variable

Using $random and $urandom

```
module tb;
  bit [31:0] a;
  bit b;
  int c;
  int d;
  initial
    begin
      a=$urandom();
      b=$urandom_range(30);
      c=$urandom_range(10,30);
      d=$random();
      $display("A=%0d B=%0d C=%0d D=%0d",a,b,c,d);
    end
endmodule
```

OUTPUT : A=2428771277 B=1 C=26 D=303379748

https://www.edaplayground.com/x/CTKh

Program 2 : Randomized a variable

Using std::randomize()

Syntax:

Std::randomize(variable)

Std::randomize(variable) with {constraints;}

```
module tb;
  bit [3:0] addr;
  bit [3:0] data;
  initial
    begin
      std::randomize(addr,data);
      $display("addr=%0d data=%0d",addr,data);
    end
endmodule
```

output : addr=13 data=2

https://www.edaplayground.com/x/Fh_2

Program 3 : Write a code for the below specification:

- Input variable a,b are declared in the module.
- Generate random numbers such that a>b.
- Do not use $random or $urandom

Hint : Used std::randomize(variable) with {constraint;};

```
module tb;
 bit [7:0] a; //variable declaration
 bit [7:0] b;
 initial
  begin
   repeat(3)
    begin
     std::randomize(a,b) with {a>b;};
   $display("a=%0d b=%0d",a,b);
    end
  end
endmodule
```

output :
```
# KERNEL: a=233 b=174
# KERNEL: a=35 b=12
# KERNEL: a=155 b=124
```

https://edaplayground.com/x/FDJ5

Program 4 : Generating random value without using constraints

Hint: min+{random} %(max-min)

```
module tb();
 integer a;
 initial
  begin
   repeat(5)
    begin
     a= 10 +{$random} % (20-10);
     $display("a=%0d",a);
    end
  end
   endmodule
```

```
output : # KERNEL: a=18
         # KERNEL: a=17
         # KERNEL: a=17
         # KERNEL: a=17
         # KERNEL: a=17
```

https://edaplayground.com/x/Zcrd

How you debugged randomization failure.

1)Check initialization of variable

2) Check constraints

3)Add debug statements: $display or $monitor

4) Enable randomization tracing

Is the randomize() method virtual?

1) NO,it is not virtual()
2) It is built in method provided by system Verilog
3) It is used for randomization of objects based on their constraints and randomization methods

Is it possible to generate 64 bit random value in verilog?

We can do this by concatenating two 32 bit to get 64 bit random values

Hint

Random64={random2,random1}-remainder

```
module tb;
  reg [31:0] random1;
  reg [31:0] random2;
  reg [63:0] random64;
  integer remainder;
  initial
   begin
    random1=$random;
    random2=$random;
    remainder={random2,random1}%10;
    random64 = {random2,random1}-remainder;
    $display("random64=%0h",random64);
    $display("size of random64",$size(random64));
   end
 endmodule
```

```
output: # KERNEL: random64=c0895e8112153524
        # KERNEL: size of random64  is 64
```

https://edaplayground.com/x/RXAr