```
// Session 15 Assignment


// Task 1


scala> def calculateGCD(x:Int,y:Int):Int={   //  func to calculate GCD of
two numbers
     | val a=x%y
     | if(a==0)
     | y
     | else
     | calculateGCD(y,a)
     | }
calculateGCD: (x: Int, y: Int)Int

scala> def determineGCD(a:Int,b:Int):Int={  // func to determine gcd of
two number using recursion
     | if(a>b)
     | calculateGCD(a,b)
     | else
     | calculateGCD(b,a)
     | }
determineGCD: (a: Int, b: Int)Int


// output

scala> println(determineGCD(20,30))   // calling function to find gcd
10

scala> println(determineGCD(20,28))   // calling function to find gcd
4

scala> println(determineGCD(98,56))   // calling function to find gcd
14



// Task 2

Part 1
// determing nth digit of fibbonaci using for loop


scala> var ini=1;var prev=0;var next=0;
ini: Int = 1
prev: Int = 0
next: Int = 0

scala> def generateFibForLoop()={
     | next=ini+prev
     | prev=ini
     | ini=next
     | }
generateFibForLoop: ()Unit


// output

scala> var n=4
n: Int = 4

scala> println(s"Suppose digit to find is $n")
Suppose digit to find is 4


scala> for(i<-1 until n)
     | {
     | generateFibForLoop()
     | if(i==n-1)
     | println(s"$n digit os fibnnnoci series is $next")
     | }
4 digit os fibnnnoci series is 3
```

Part2

```
// determing nth digit of fibonnaci using recursion

scala> var next=0;
next: Int = 0

                          ^

scala> def generateFibRec(x:Int, y:Int,index:Int):Int={
     | var ini=x;var prev=y;
     | if(index>1)
     | {
     | next=ini+prev
     | prev=ini
     | ini=next
     | generateFibRec(ini,prev,index-1)
     | }
     | next
     | }
generateFibRec: (x: Int, y: Int, index: Int)Int

scala> println(s"Suppose 4th digit to be find")
Suppose 4th digit to be find


//output:-
```

```
scala> println(s"4 th digit of the fibbonaci is
${generateFibRec(1,0,4)}")
4 th digit of the fibbonaci is 3



// Task 3

// determining the square root of a number.

scala> def check(iniApprox:Double,nextApprox:Double):Int={
     | val a=(iniApprox*1000).round/1000.toDouble
     | val b=(nextApprox*1000).round/1000.toDouble
     | if(a==b)
     | 1
     | else
     | 0
     | }
check: (iniApprox: Double, nextApprox: Double)Int


scala> var number=16
number: Int = 16


scala> def getSquareRoot(iniApprox:Double):Double={
     | val fac=number.toDouble/iniApprox
     | val nextApprox=(iniApprox+fac)/2.toDouble
     | if(check(iniApprox,nextApprox)==1)
     | nextApprox
     | else
     | getSquareRoot(nextApprox)
     | }
getSquareRoot: (iniApprox: Double)Double


// output

scala> number
res0: Int = 16

scala> getSquareRoot(8)
res1: Double = 4.000000000000004

scala> println("suppose initial approx for number 16 is 8")
suppose initial approx for number 16 is 8
```