ASSIGNMENT SCALA 2

Problem Statement

Task 1:

Create a Scala application to find the GCD of two numbers
Solution:

EXPLANATION:
To find the GCD of two numbers I have used the below logic:

A) If either 1st or 2nd number is 0, then other number is the Greatest Common Divisor

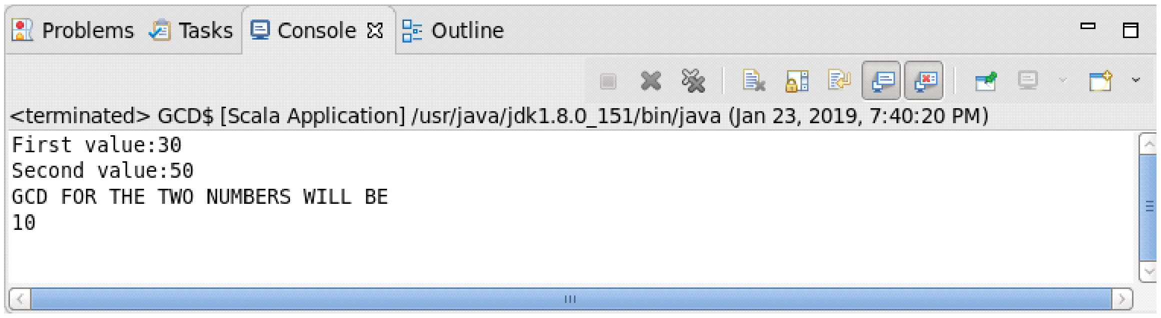
B) Else call the GCD function again by sending 2nd number as 1st number and difference between 2 numbers as 2nd number.

C) This in turn checks for the If clause again.

SOLUTION REPORT

```
package scala_assignment

object GCD {
//gcd function
  def gcd(a: Int,b: Int): Int = {
       if(b ==0) a

       else gcd(b, a%b)
    }

  def main(args: Array[String]) {
//Initialising Two values
      var val1 = 30
      var val2 =50
      println("First value:"+val1)
      println("Second value:"+val2)
      println("GCD FOR THE TWO NUMBERS WILL BE")

//Calling gcd function

      println(gcd(val1,val2))

    }
}
```

OUTPUT:



```
<terminated> GCD$ [Scala Application] /usr/java/jdk1.8.0_151/bin/java (Jan 23, 2019, 7:40:20 PM)
First value:30
Second value:50
GCD FOR THE TWO NUMBERS WILL BE
10
```

TASK 2:

Fibonacci series (starting from 1) written in order without any spaces in between, thus producing a sequence of digits.

Write a Scala application to find the Nth digit in the sequence.
A) Write the function using standard for loop
B) Write the function using recursion

A) FIBONACCI SERIES USING LOOPS:-

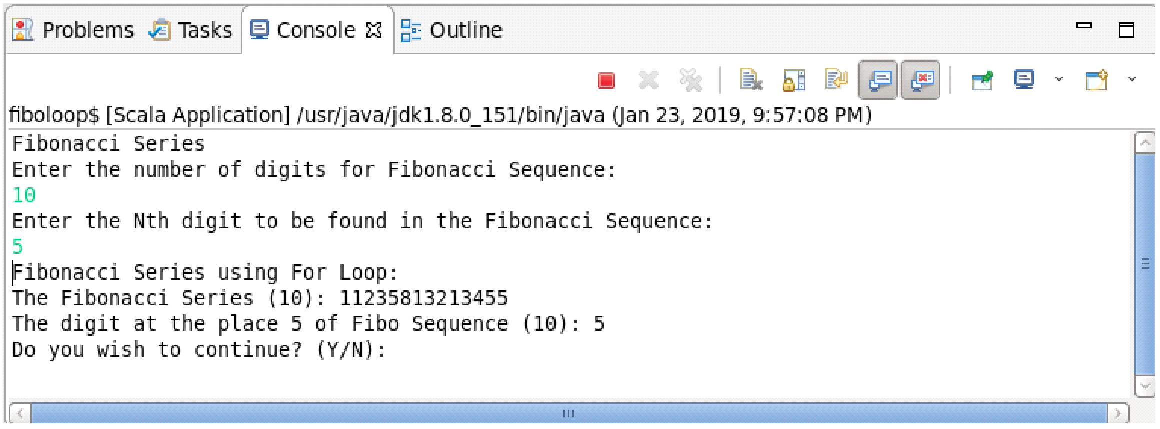
EXPLANATION: To find the Fibonacci Series using a Standard FOR Loop. This is achieved by the method in which I created a function "LoopFibo(digits, nthdigit)".

SOLUTION REPORT:

```
package scala_assignment

object fiboloop {
  def LoopFibo(n: Int, nth: Int): Unit = {
    var concat_result = "1"

    if (n < 2) {
      println(n)
    }

    else {
      var result: BigInt = 0
      var n1: BigInt = 0
      var n2: BigInt = 1

      for (i <- 1 until n) {
        result = n1 + n2
        n1 = n2
        n2 = result
        concat_result = concat_result + result
      }

      get_nthchar_and_print(n, concat_result, nth)
```

```
        result
      }
    }

//Displaying Nth character in the Fibonacci Sequence***
    def get_nthchar_and_print(n: Int, seq: String, nth: Int): Unit = {
        println(s"The Fibonacci Series ($n): " + seq)
        println(s"The digit at the place $nth of Fibo Sequence ($n): " +
seq.charAt(nth -1).toChar)
    }

    def main(args: Array[String]): Unit = {

        var repeat = " "
        println("Fibonacci Series")
    do
    {
        println("Enter the number of digits for Fibonacci Sequence:")
        var digits: Int = scala.io.StdIn.readLine().toInt

        println("Enter the Nth digit to be found in the Fibonacci Sequence:")
        var nthFind: Int = scala.io.StdIn.readLine().toInt

//Calling function "LoopFibo" to find out the Fibonacci Series using For
Loop
        println(s"Fibonacci Series using For Loop:")
        LoopFibo(digits, nthFind)

//Do-While Loop for continuing the process
        println("Do you wish to continue? (Y/N):")
        repeat = scala.io.StdIn.readLine().toUpperCase
   }
        while (repeat.equals("Y"))
   }
}
```

OUTPUT:



```
fiboloop$ [Scala Application] /usr/java/jdk1.8.0_151/bin/java (Jan 23, 2019, 9:57:08 PM)
Fibonacci Series
Enter the number of digits for Fibonacci Sequence:
10
Enter the Nth digit to be found in the Fibonacci Sequence:
5
Fibonacci Series using For Loop:
The Fibonacci Series (10): 11235813213455
The digit at the place 5 of Fibo Sequence (10): 5
Do you wish to continue? (Y/N):
```

B) FIBONACCI SERIES USING RECURSION:-

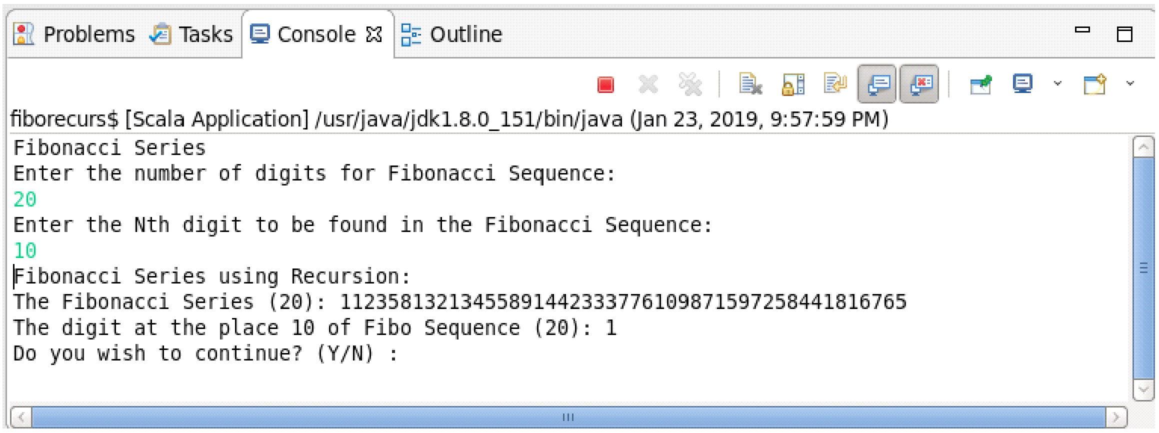
EXPLANATION: Using Recursion in which I made a fucntion "recFibonacci(digits,nthdigit)". The "@tailrec annotation" in the code is used to indicate that this is an optimized version of the function to find the Fibonacci series as a recursive function is tail recursive when the recursive call is the last thing executed by the function.

SOLUTION REPORT

```
package scala_assignment

import scala.annotation.tailrec

object fiborecurs {

    def recFibonacci(n: Int, nth: Int): Unit = {
    var concat_result = "1"

//Method to find out the Fibonacci Series using Recursion
@tailrec def fiboRecursive(n: Int, prev: BigInt = 0, next: BigInt = 1):
BigInt = n match {

      case 0 => prev
      case 1 => next
      case _ =>
        concat_result = concat_result + (prev + next)
        fiboRecursive(n - 1, next, next + prev)
    }

    fiboRecursive(n)
    get_nthchar_and_print(n, concat_result, nth)
  }

//Method to display Nth character in the Fibonacci Sequence
  def get_nthchar_and_print(n: Int, seq: String, nth: Int): Unit = {
      println(s"The Fibonacci Series ($n): " + seq)
      println(s"The digit at the place $nth of Fibo Sequence ($n): " +
seq.charAt(nth -1).toChar)
  }

  def main(args: Array[String]): Unit = {
      var repeat = " "

      println("Fibonacci Series")

  do
  {
      println("Enter the number of digits for Fibonacci Sequence: ")
      var digits: Int = scala.io.StdIn.readLine().toInt

      println("Enter the Nth digit to be found in the Fibonacci Sequence:
")
```

```
      var nthFind: Int = scala.io.StdIn.readLine().toInt

//Calling function "recFibonacci" to find out the Fibonacci Series using
Recursion

      println(s"Fibonacci Series using Recursion:")
      recFibonacci(digits, nthFind)
      println("Do you wish to continue? (Y/N) : ")

//Do-While Loop for continuing the process
      repeat= scala.io.StdIn.readLine().toUpperCase
  }

      while (repeat.equals("Y"))
  }
}
```

OUTPUT:



```
fiborecurs$ [Scala Application] /usr/java/jdk1.8.0_151/bin/java (Jan 23, 2019, 9:57:59 PM)
Fibonacci Series
Enter the number of digits for Fibonacci Sequence:
20
Enter the Nth digit to be found in the Fibonacci Sequence:
10
Fibonacci Series using Recursion:
The Fibonacci Series (20): 11235813213455891442333776109871597258441816765
The digit at the place 10 of Fibo Sequence (20): 1
Do you wish to continue? (Y/N) :
```

TASK 3

Find square root of number using Babylonian method.
A) Start with an arbitrary positive start value x (the closer to the root, the better).
B) Initialize y = 1.

Do following until desired approximation is achieved.

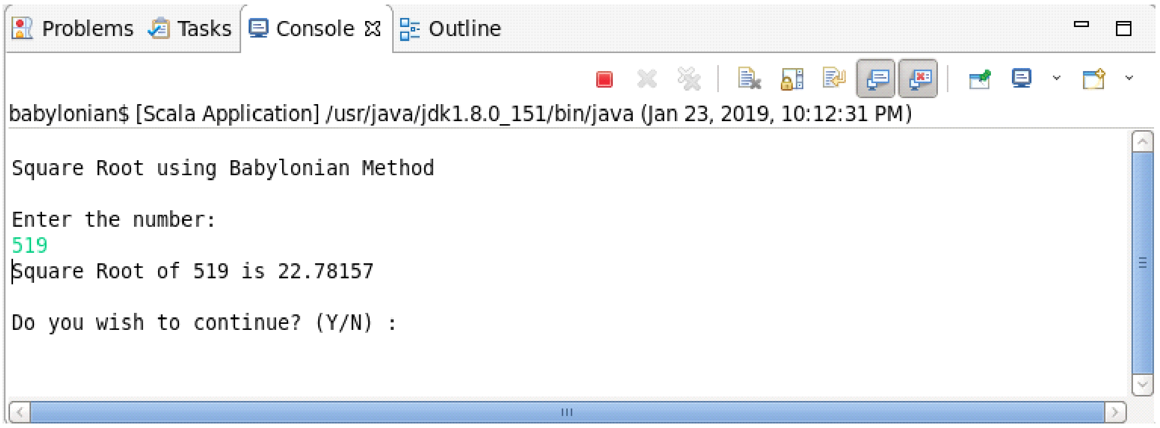
C) Get the next approximation for root using average of x and y
D) Set y = n/x

EXPLANATION: The Babylonian method for finding square roots involves dividing and averaging, over and over, to obtain a more accurate solution with each repeat of the process.

SOLUTION REPORT:

```
package scala_assignment

object babylonian {
//Function to return square root of a number using Babylonian Method

  def squareRootBM(num: Int): Float = {

//Arbitrary positive value x from the user

    var x: Float = num

//Initialize y

    var y: Float = 1

//e decides the accuracy level(checked when we aren't sure if the number
is a perfect square)

    val e: Double = 0.000001

//Performing division and averaging until the accuracy level

    while(x - y > e)
    {
      x = (x + y) / 2
      y = num / x
    }
    x //Returns the square root value
  }

  def main(args: Array[String]): Unit = {
    var continue = " "

    println("\nSquare Root using Babylonian Method")

    do
    {
      println("\nEnter the number: ")
      var input = scala.io.StdIn.readLine().toInt

//Calling the function to calculate Square Root using Babylonian Method

      println(s"Square Root of $input is ${squareRootBM(input)}")

      println("\nDo you wish to continue? (Y/N) : ")

//Do-While Loop for continuing the process

      continue = scala.io.StdIn.readLine().toUpperCase
    }
    while (continue.equals("Y"))
  }
}
```

OUTPUT:



babylonian$ [Scala Application] /usr/java/jdk1.8.0_151/bin/java (Jan 23, 2019, 10:12:31 PM)

```
Square Root using Babylonian Method

Enter the number:
519
Square Root of 519 is 22.78157

Do you wish to continue? (Y/N) :
```