BIG DATA HADOOP & SPARK TRAINING

Assignment-15: Assignment on Scala II

APRIL 23, 2018

RASHMI KRISHNA

Task 1

Create a Scala application to find the GCD of two numbers

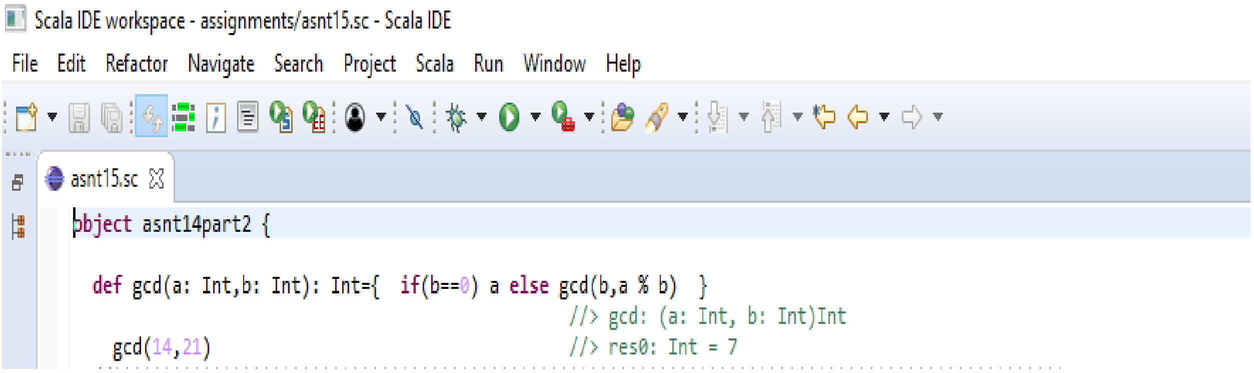
```
object asnt14part2 {

 def gcd(a: Int,b: Int): Int={  if(b==0) a else gcd(b,a % b)  }

  gcd(14,21)
```

The above program shows the GCD program:

- Here function gcd is taking two parameters i.e a and b which are Integers.
- In the body of this function, we are checking if second number i.e. b is zero or not.
  - If “b” is zero then gcd is same value as that of variable “a”
  - If “b” is not zero then gcd of two numbers is found by b and a modulus of b
- We can see how the program is implemented and what is the output we have received i.e. gcd of (14,21) is 7



```
object asnt14part2 {

  def gcd(a: Int,b: Int): Int={  if(b==0) a else gcd(b,a % b)  }
                                                  //> gcd: (a: Int, b: Int)Int
   gcd(14,21)                                     //> res0: Int = 7
```

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.

➢ Write the function using standard for loop

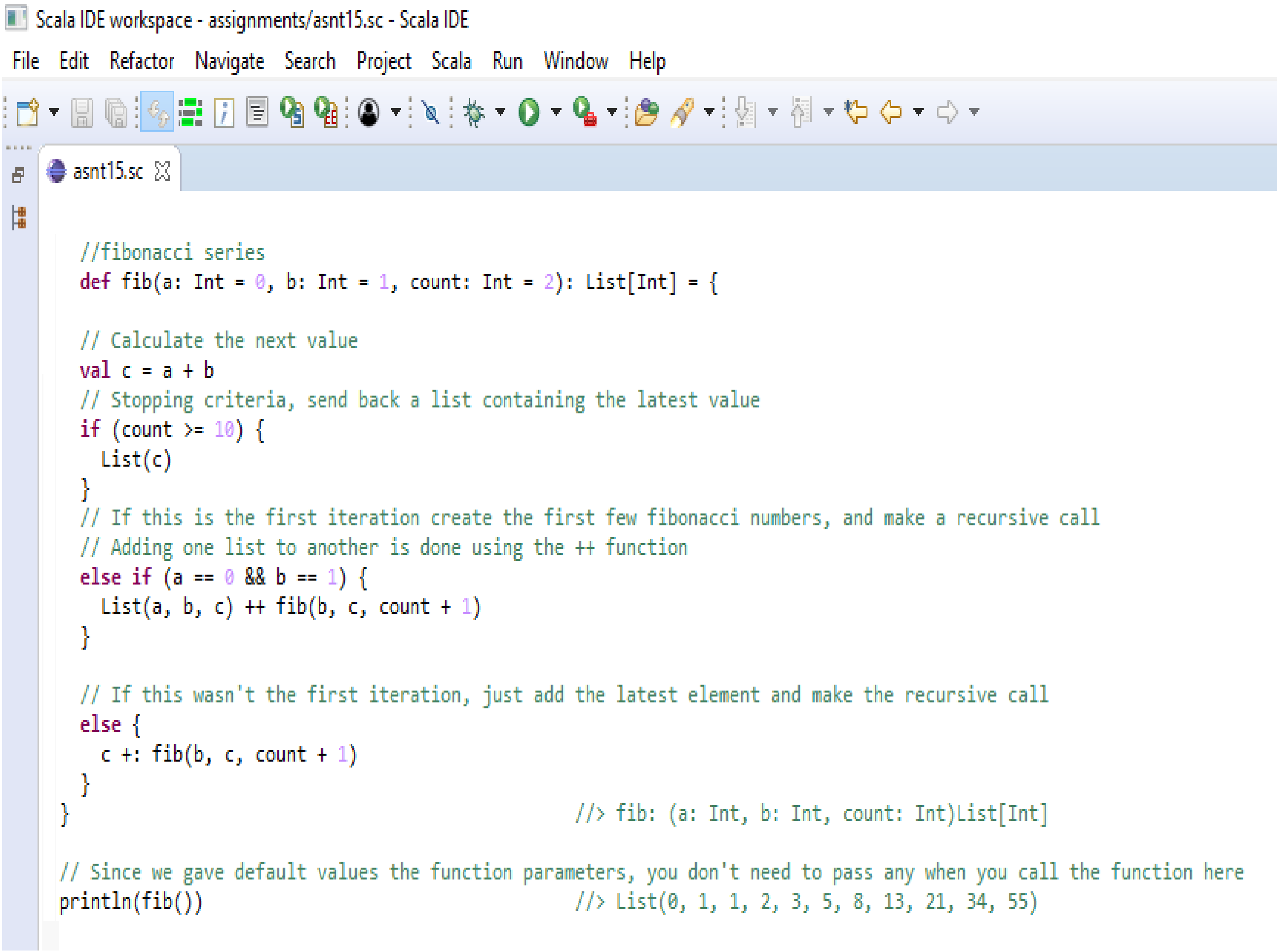
➢ Write the function using recursion

```
 def fib(a: Int = 0, b: Int = 1, count: Int = 2): List[Int] = {

 // To calculate the next value we add first and second number
 val c = a + b
 // Stopping criteria, send back a list containing the latest value
 if (count >= 10) {
  List(c)
 }
 // If this is the first iteration create the first few fibonacci numbers, and make a recursive call
 // Adding one list to another is done using the ++ function
 else if (a == 0 && b == 1) {
  List(a, b, c) ++ fib(b, c, count + 1)
 }

 // If this wasn't the first iteration, just add the latest element and make the recursive call
 else {
  c +: fib(b, c, count + 1)
 }
}

// Since we gave default values the function parameters, you don't need to pass any when you call the function here
println(fib())
```



```
  //fibonacci series
  def fib(a: Int = 0, b: Int = 1, count: Int = 2): List[Int] = {

  // Calculate the next value
  val c = a + b
  // Stopping criteria, send back a list containing the latest value
  if (count >= 10) {
    List(c)
  }
  // If this is the first iteration create the first few fibonacci numbers, and make a recursive call
  // Adding one list to another is done using the ++ function
  else if (a == 0 && b == 1) {
    List(a, b, c) ++ fib(b, c, count + 1)
  }

  // If this wasn't the first iteration, just add the latest element and make the recursive call
  else {
    c +: fib(b, c, count + 1)
  }
}                                             //> fib: (a: Int, b: Int, count: Int)List[Int]

// Since we gave default values the function parameters, you don't need to pass any when you call the function here
println(fib())                                //> List(0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55)
```

Task 3

Find square root of number using Babylonian method.

1. Start with an arbitrary positive start value x (the closer to the root, the better).

2.Initialize y = 1.

3. Do following until desired approximation is achieved.

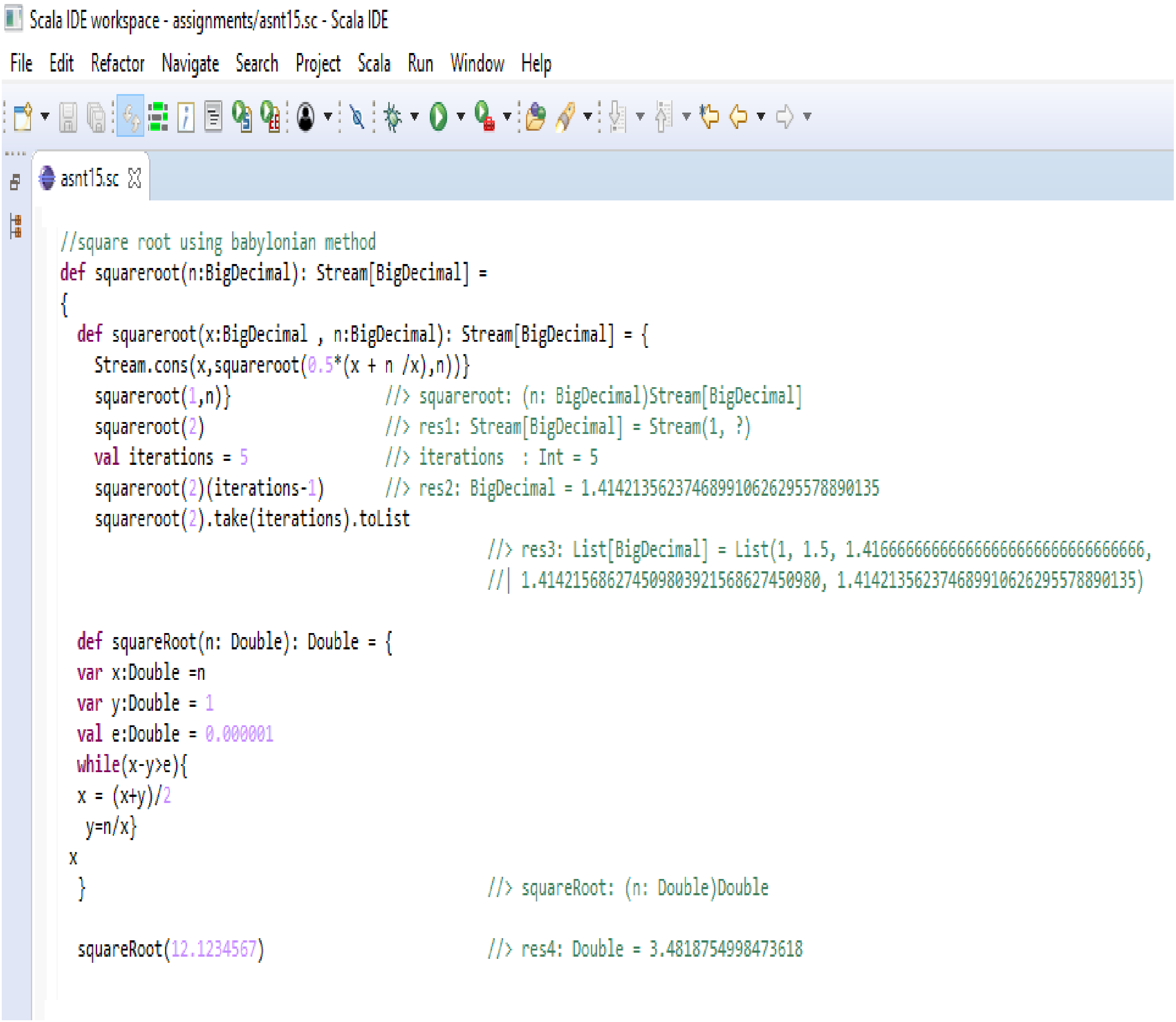
a) Get the next approximation for root using average of x and y

b) Set y = n/x

```
// squareroot function to find the square root of a number using Babylonian method

def squareroot(n:BigDecimal): Stream[BigDecimal] =
{
                    def squareroot(x:BigDecimal , n:BigDecimal): Stream[BigDecimal] = {
                    Stream.cons(x,squareroot(0.5*(x + n /x),n))}
// to find squareroot of x, we will add x with n(root number) and multiply the result //with
0.5
                    squareroot(1,n)}
                    squareroot(2                    // streaming 5 iterations to find squareroot
                    val iterations = 5
                    squareroot(2)(iterations-1)
                    squareroot(2).take(iterations).toList

 def squareRoot(n: Double): Double = {
 var x:Double =n
 var y:Double = 1
 val e:Double = 0.000001
 while(x-y>e){
 x = (x+y)/2
  y=n/x}                                                  // while loop to find the squareroot
x
 }
//  Calling the "squareroot" function to execute the square root of 12.1234556
 squareRoot(12.1234567)               //output:> res4: Double = 3.4818754998473618
```



Scala IDE workspace - assignments/asnt15.sc - Scala IDE
File Edit Refactor Navigate Search Project Scala Run Window Help
asnt15.sc

```
//square root using babylonian method
def squareroot(n:BigDecimal): Stream[BigDecimal] =
{
  def squareroot(x:BigDecimal , n:BigDecimal): Stream[BigDecimal] = {
    Stream.cons(x,squareroot(0.5*(x + n /x),n))}
    squareroot(1,n)}                  //> squareroot: (n: BigDecimal)Stream[BigDecimal]
    squareroot(2)                     //> res1: Stream[BigDecimal] = Stream(1, ?)
    val iterations = 5                //> iterations  : Int = 5
    squareroot(2)(iterations-1)       //> res2: BigDecimal = 1.414213562374689910626295578890135
    squareroot(2).take(iterations).toList
                                                  //> res3: List[BigDecimal] = List(1, 1.5, 1.416666666666666666666666666666666,
                                                  //| 1.414215686274509803921568627450980, 1.414213562374689910626295578890135)

  def squareRoot(n: Double): Double = {
  var x:Double =n
  var y:Double = 1
  val e:Double = 0.000001
  while(x-y>e){
  x = (x+y)/2
   y=n/x}
 x
  }                                               //> squareRoot: (n: Double)Double

  squareRoot(12.1234567)                          //> res4: Double = 3.4818754998473618
```