databricks Assignment-44.1

```
// Databricks notebook source
// Task 1
/*
A 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
*/


// The Fibonacci sequence is one of the most famous formulas in mathematics. Each number in the sequence is the sum of the two numbers that precede it. So, the
sequence goes: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34 as so on


//Fibonacci using Loops
object fibLoop
{
    def fib( n : Int ) : Int = {
    var a = 1
    var b = 2
    println(a)

    for(i <- 1 to n-1)
    {
      val c = a + b
      a = b
      b = c
      println(a)
    }

    return a
    }


    def main(args: Array[String]) {
        var nth_element = 0
        println("Fibonacci series with total " + args(0) + " elements is (starting from 1) : ")
        nth_element = fib(args(0).toInt)
        println("element " + args(0) + " in fibonacci series is : " + nth_element)
    }
 }

defined object fibLoop

fibLoop.main(Array("10"))

Fibonacci series with total 10 elements is (starting from 1) :
1
2
3
5
8
13
21
34
55
89
element 10 in fibonacci series is : 89

//Fibonacci using Recursion


object fibRecursion
{
    def fib( n : Int) : Int = n match {
        case 1 | 2 => n
        case _ => fib(n-1) + fib(n-2)
        }

    def main(args: Array[String]) {
        println("Element " + args(0) + " in fibonacci series is (starting from 1): " + fib(args(0).toInt))
    }
 }

defined object fibRecursion

fibRecursion.main(Array("10"))

Element 10 in fibonacci series is (starting from 1): 89

//Task 2
/*
Create a calculator to work with rational numbers. Requirements:
- 2.1 It should provide capability to add, subtract, divide and multiply rational numbers
- 2.2 Create a method to compute GCD (this will come in handy during operations on rational)
Add option to work with whole numbers which are also rational numbers i.e. (n/1)
- achieve the above using auxiliary constructors
- enable method overloading to enable each function to work with numbers and rational.
*/
```

```
//Scala Class - Rational Numbers
class Rational(n: Int, d: Int)
{
  require(d != 0)
  override def toString = n +"/"+ d
  private def gcd(x: Int, y: Int): Int =
  {
    if (x == 0) y
    else if (x < 0) gcd(-x, y)
    else if (y < 0) -gcd(x, -y)
    else gcd(y % x, x)
  }

  private val g = gcd(n, d)
  val numer: Int = n/g
  val denom: Int = d/g


  // Add (+) method
  def +(that: Rational) = new Rational(numer * that.denom + that.numer * denom, denom * that.denom)

  // Subtract (-) method
  def -(that: Rational) = new Rational(numer * that.denom - that.numer * denom, denom * that.denom)

  // Multiply (*) method
  def *(that: Rational) = new Rational(numer * that.numer, denom * that.denom)

  // Divide (/) method
  def /(that: Rational) = new Rational(numer * that.denom, denom * that.numer)

  // Print Method
  def printRat():Float = { var res:Float = 0; res = numer/denom; return(res) }
}

defined class Rational

var a = new Rational(7,0)

  java.lang.IllegalArgumentException: requirement failed

var a = new Rational(5,7)
var b = new Rational(3,8)

a: Rational = 5/7
b: Rational = 3/8

//a.printRat()


//b.printRat()


var c= a + b

c: Rational = 61/56

var c= a - b

c: Rational = 19/56

var c= a * b

c: Rational = 15/56

var c= a / b

c: Rational = 40/21

//Task 3

// 3.1.Write a simple program to show inheritance in scala.


//Scala - Inheritance
//Parent Class
class Employee
{
  var salary:Float = 10000
}
//Child Class
class Programmer extends Employee
{
  var bonus:Int = 5000
  println("Salary = " + salary)
  println("Bonus = " + bonus)
}
//Create Object

new Programmer()

Salary = 10000.0
Bonus = 5000
defined class Employee
defined class Programmer
res47: Programmer = Programmer@4851279d

// 3.2.Write a simple program to show multiple inheritance in scala.
```

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

    trait A {
      var distance: Int = _
      def action = {
        distance = distance + 5
        println("A - Distance  :  "+ distance)
      }
    }

    trait B {
      var driverVar: Int = _
      def action = {
        driverVar = driverVar + 1
        println("B - Driver   :  "+ driverVar)
      }
    }

    class AB extends A with B {
     distance = 3;
     driverVar = 6;
     println("Drive    - AB :  "+ driverVar)
     println("Distance - AB :  "+ distance)
     override def action = {
     println("Super A - Enter")
     super[A].action
     println("Super A - Exit")
     println("Super B - Enter")
     super[B].action
     println("Super B - Exit")
     }
    }

    var ab = new AB
    ab.action
//    println("Drive    :  "+ ab.driverVar)
//    println("Distance :  "+ ab.distance)
  }
}

defined object Simulation

Simulation.main(Array())

Drive    - AB :  6
Distance - AB :  3
Super A - Enter
A - Distance  :  8
Super A - Exit
Super B - Enter
B - Driver   :  7
Super B - Exit

// 3.3 Write a partial function to add three numbers in which one number is constant and two numbers can be passed as inputs and define another method which
can take the partial function as input and squares the result.


// The following partial function adds 12 to each number in the tuple passed to it
val addConstantTo: PartialFunction[(Int, Int), Int] = {
   case (a, b) => a + b + 12
}

addConstantTo: PartialFunction[(Int, Int),Int] = <function1>

addConstantTo((12, 34))

res53: Int = 58

// 3.4.Write a program to print the prices of 4 courses of Acadgild: Android-12999,Big Data Development-17999,Big Data Development-17999,Spark-19999 using
match and add a default condition if the user enters any other course


// Scala function - Print the prices of courses of AcadGild
object CourseSelection
{
  def matchCourse(x:String): String =
  {
      x match
        {
          case "Android" => "Android-12999"
          case "BigData" => "BigDataDevelopment-17999"
          case "Spark" => "Spark-19999"
          case _ => "Android-12999"
        }
  }
  def main(args: Array[String])
  {
      // val course = new CourseSelection()
      println(matchCourse(args(0)))
  }
}


defined object CourseSelection

CourseSelection.main(Array("XYZ"))

Android-12999
```

```
CourseSelection.main(Array("Android"))
```

```
Android-12999
```

```
CourseSelection.main(Array("BigData"))
```

```
BigDataDevelopment-17999
```

```
Spark-19999
```