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**

fibonacci_for_loop.scala

```
object fibonacci_for_loop
{
        def fib(n: Int) : Int =
        {
                var a = 0
                var b = 1
                var i = 0

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

        def main(args: Array[String]){
                println(fib(10))
        }
}
```

acadgild@localhost:~

```
[acadgild@localhost ~]$ scalac fibonacci_for_loop.scala
[acadgild@localhost ~]$ scala fibonacci_for_loop
55
[acadgild@localhost ~]$
```

Applications Places System

Browse and run installed applications

fibonacci_recursion.scala (~) - gedit

File Edit View Search Tools Documents Help

Open Save Undo

fibonacci_recursion.scala

```
object fibonacci_recursion
{
        def fib_recursion(n: Int) : Int = n match
        {
                case 0 | 1 => n
                case _ => fib_recursion(n-1) + fib_recursion(n-2)
        }

        def main(args: Array[String]){
                println(fib_recursion(10))
        }
}
```

```
[acadgild@localhost ~]$ scalac fibonacci_recursion.scala
[acadgild@localhost ~]$ scala fibonacci_recursion
55
[acadgild@localhost ~]$
```

Task 2

Create a calculator to work with rational numbers.

Requirements:

○ It should provide capability to add, subtract, divide and multiply rational numbers

○ 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.

```
class Rational (n: Int, d: Int)
{
        /* auxiliary constructors */

        def this(n: Int) = this(n, 1)

        /* compute GCD */
        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

        def +(r: Rational) =
                new Rational(numer * r.denom + r.numer * denom,
                             denom * r.denom)

        def -(r: Rational) =
                new Rational(numer * r.denom - r.numer * denom,
                            denom * r.denom)

        def *(r: Rational) =
                new Rational(numer * r.numer, denom * r.denom)

        def /(r: Rational) =
                new Rational(numer * r.denom, denom * r.numer)
}
```

```
bject rational_Calculator {
        def main(args: Array[String])
        {
                /* add three numbers or rational numbers */

                var x = new Rational(2, 3)
                var y = new Rational(2, 1)
                var z = new Rational(3, 2)

                val output = x + y + z
                println("" + output.numer + "/" + output.denom)
        }
}
```

```
[acadgild@localhost ~]$ scalac rational_Calculator.scala
[acadgild@localhost ~]$ scala rational_Calculator
25/6
[acadgild@localhost ~]$
```

Task 3

1.Write a simple program to show inheritance in scala.

2.Write a simple program to show multiple inheritance in scala.

Applications Places System

simple_inheritance.scala (~) - gedit

File Edit View Search Tools Documents Help

Open Save Undo

simple_inheritance.scala

```
class Employee {
        var salary:Float = 10000
}

class Programmer extends Employee {
        var bonus:Int = 5000
        println("Salary = "+ salary)
        println("Bonus = "+ bonus)
}

object simple_inheritance {
        def main(args:Array[String]){
                new Programmer()
        }
}
```

Applications Places System

acadgild@localhost:~

File Edit View Search Terminal Help

```
[acadgild@localhost ~]$ scalac simple_inheritance.scala
[acadgild@localhost ~]$ scala simple_inheritance
Salary = 10000.0
Bonus = 5000
[acadgild@localhost ~]$
```

Applications Places System

multi_inheritance.scala (~) - gedit

File Edit View Search Tools Documents Help

Open Save Undo

multi_inheritance.scala

```
class A {
        var salary1 = 10000
}

class B extends A {
        var salary2 = 20000
}

class C extends B {
        def show() {
                println("Salary1 = " + salary1)
                println("Salary2 = " + salary2)
        }
}

object multi_inheritance
{
        def main(args:Array[String]){
                var c = new C()
                c.show()
        }
}
```

Applications Places System

acadgild@localhost:~

File Edit View Search Terminal Help

```
[acadgild@localhost ~]$ scalac multi_inheritance.scala
[acadgild@localhost ~]$ scala multi_inheritance
Salary1 = 10000
Salary2 = 20000
[acadgild@localhost ~]$
```

Task 4. 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.

partial_function.scala (~) - gedit

```
class Function1
{
        val add = (x: Int, y: Int) => 10 + x + y // here x and y are inputs to the partial function add and 10 is the constant

        def square(add: Int) :Int = add*add // square functon takes add as the input
}

object partial_function
{
        def main(args: Array[String]){
          val a = new Function1()
          var b = a.add(5, 6)
          println("sum of three numbers = " + b)
          println("square of partial function result = " + a.square(b))
        }
}
```

acadgild@localhost:~

```
[acadgild@localhost ~]$ scalac partial_function.scala
[acadgild@localhost ~]$ scala partial_function
sum of three numbers = 21
square of partial function result = 441
[acadgild@localhost ~]$
```

Task 5 : 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

Applications Places System

Browse and run installed applications

Courses.scala (~) - gedit

File Edit View Search Tools Documents Help

Open Save Undo

Courses.scala

```
object Courses
{
        def main(args: Array[String]){
          val courses: Map[String, Int] =
                Map(
                        "Android" -> 12999,
                        "Big Data Development" -> 17999,
                        "Spark" -> 19999
                   )
                println(courses)

                println("Enter the course to find its price: ")
                var a :String = scala.io.StdIn.readLine()

                a match{
                        case "Android" => println(12999)
                        case "Big Data Development" => println(17999)
                        case "Spark" => println(19999)
                        case _ => println("No such course")
                }
        }
}
```

acadgild@localhost:~

File Edit View Search Terminal Help

```
[acadgild@localhost ~]$ scalac Courses.scala
[acadgild@localhost ~]$ scala Courses
Map(Android -> 12999, Big Data Development -> 17999, Spark -> 19999)
Enter the course to find its price:
Android
12999
[acadgild@localhost ~]$
```