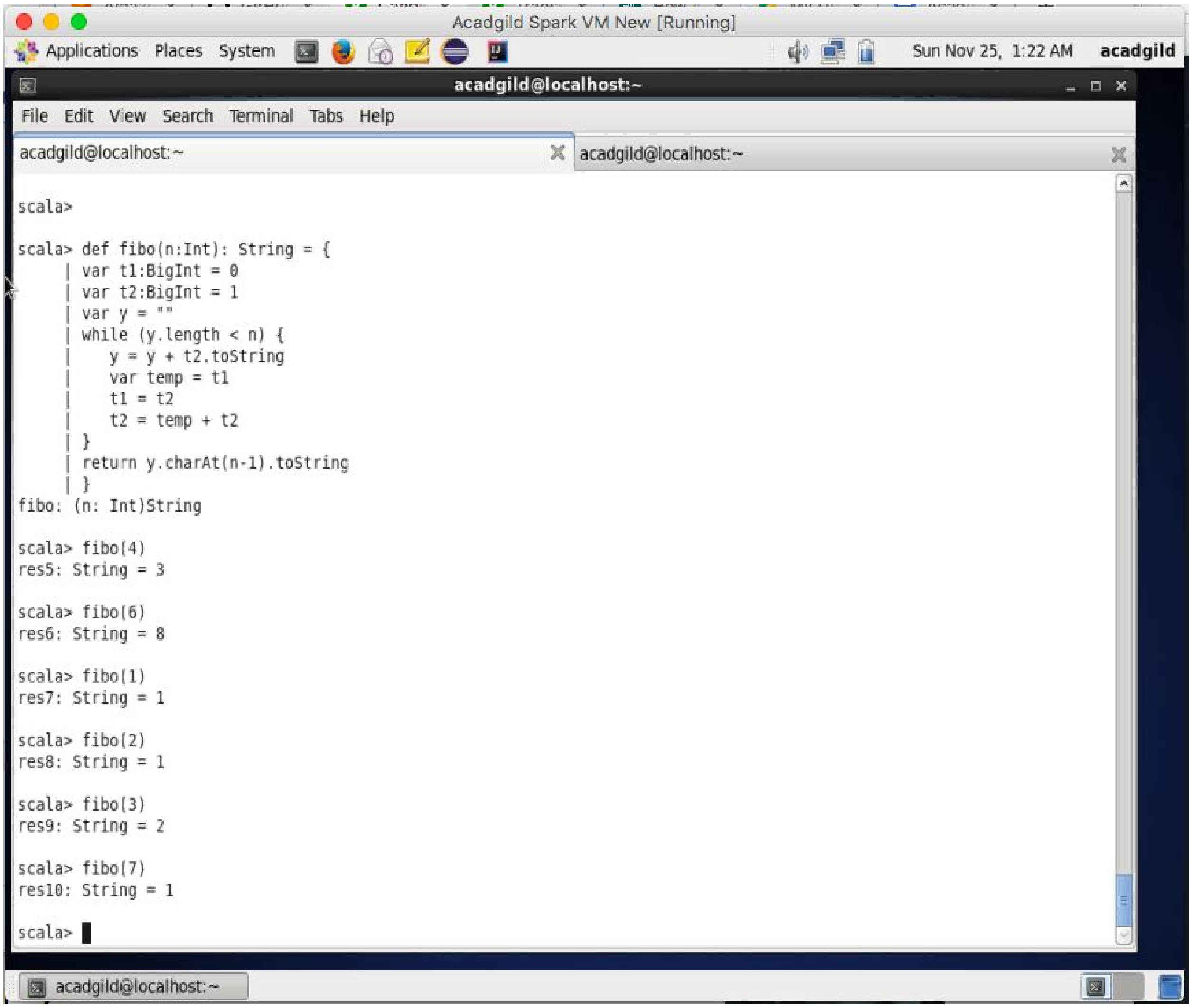
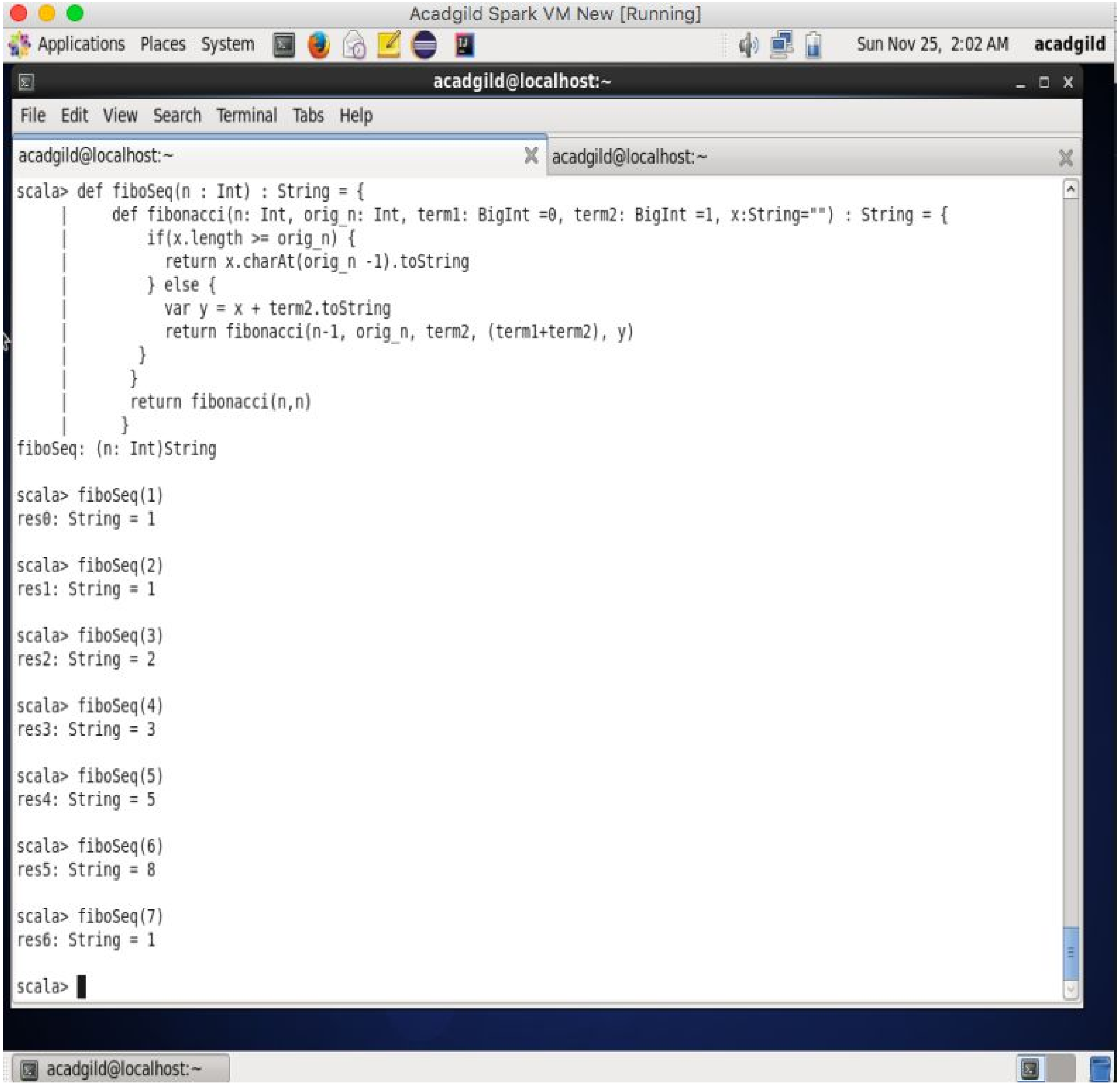
Task 1
A Fibonacci series (starting from 1) written in order without any spaces in between, thus producing a sequence of digits.


Acadgild Spark VM New [Running]
Applications Places System
Sun Nov 25, 1:22 AM acadgild
acadgild@localhost:~
File Edit View Search Terminal Tabs Help
acadgild@localhost:~
acadgild@localhost:~
acadgild@localhost:~

```
scala>

scala> def fibo(n:Int): String = {
     | var t1:BigInt = 0
     | var t2:BigInt = 1
     | var y = ""
     | while (y.length < n) {
     |    y = y + t2.toString
     |    var temp = t1
     |    t1 = t2
     |    t2 = temp + t2
     | }
     | return y.charAt(n-1).toString
     | }
fibo: (n: Int)String

scala> fibo(4)
res5: String = 3

scala> fibo(6)
res6: String = 8

scala> fibo(1)
res7: String = 1

scala> fibo(2)
res8: String = 1

scala> fibo(3)
res9: String = 2

scala> fibo(7)
res10: String = 1

scala> 
```

B: Write a Scala application to find the Nth digit in the sequence. Write the function using using recursion


Acadgild Spark VM New [Running]
Applications Places System
Sun Nov 25, 2:02 AM acadgild
acadgild@localhost:~
File Edit View Search Terminal Tabs Help
acadgild@localhost:~
acadgild@localhost:~
acadgild@localhost:~

```
scala> def fiboSeq(n : Int) : String = {
     |     def fibonacci(n: Int, orig_n: Int, term1: BigInt =0, term2: BigInt =1, x:String="") : String = {
     |        if(x.length >= orig_n) {
     |          return x.charAt(orig_n -1).toString
     |        } else {
     |          var y = x + term2.toString
     |          return fibonacci(n-1, orig_n, term2, (term1+term2), y)
     |       }
     |      }
     |      return fibonacci(n,n)
     |     }
fiboSeq: (n: Int)String

scala> fiboSeq(1)
res0: String = 1

scala> fiboSeq(2)
res1: String = 1

scala> fiboSeq(3)
res2: String = 2

scala> fiboSeq(4)
res3: String = 3

scala> fiboSeq(5)
res4: String = 5

scala> fiboSeq(6)
res5: String = 8

scala> fiboSeq(7)
res6: String = 1

scala> 
```

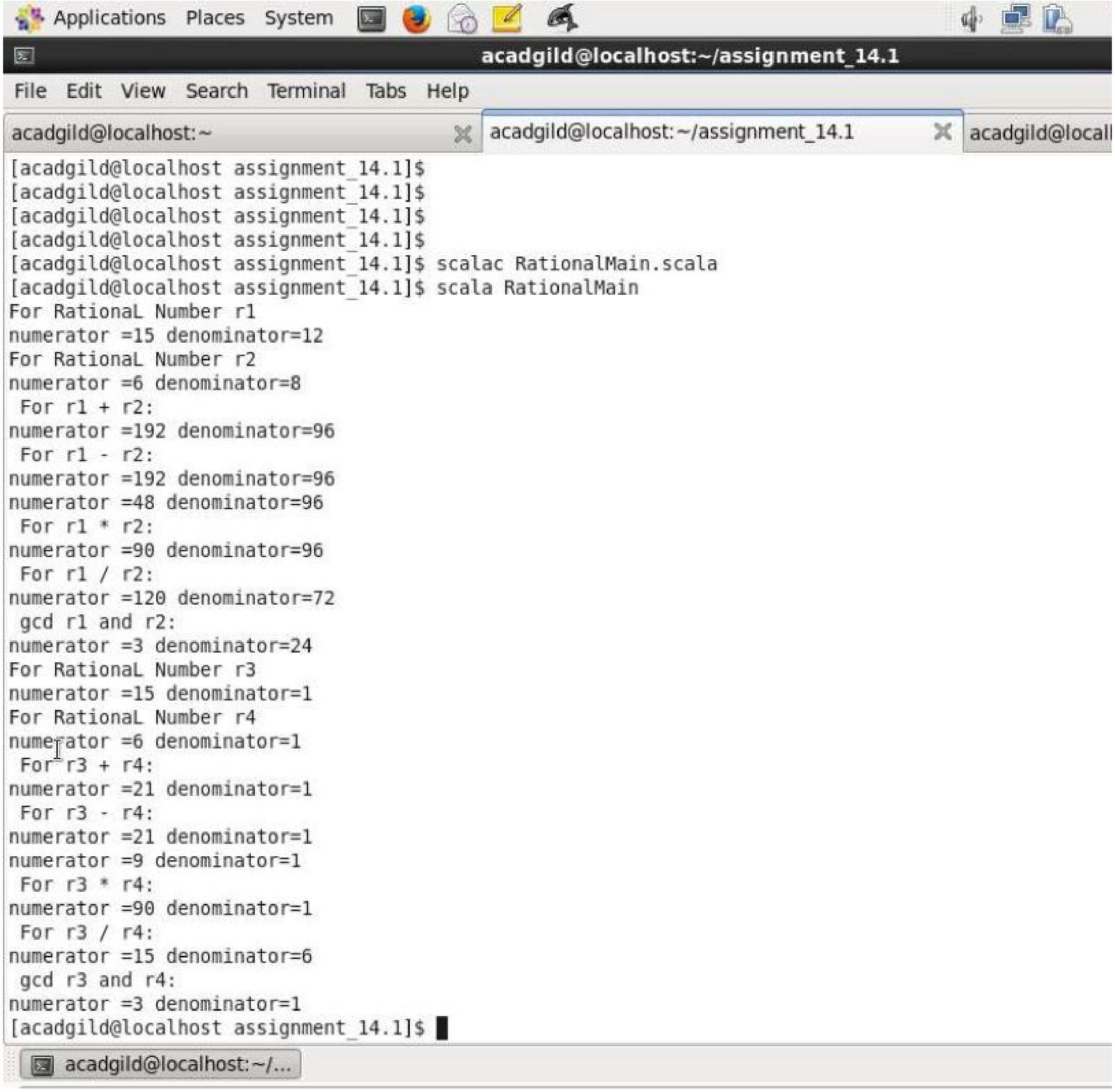
Task 2:
Create a calculator to work with rational numbers.
Requirements:
○ It should provide capability to add, subtract, divide and multiply rational numbers
○ Create a method to compute GCD (this will come in handy during operations on rational)

RationalMain.scala:

```
class Rational (x:BigInt, y:BigInt) {
private val numerator:BigInt = x
private val denominator:BigInt = y
def this(a:BigInt) = this(a, 1)
def sum(b: Rational):Rational = {
return new Rational(numerator * b.denominator + denominator * b.numerator, denominator * b.denominator)
}
def sum(b: BigInt):Rational = {
return new Rational(numerator + b, 1)
}
def subtract(b: Rational):Rational = {
return new Rational(numerator * b.denominator - denominator * b.numerator, denominator * b.denominator)
}
def subtract(b: BigInt):Rational = {
return new Rational(numerator - b, 1)
}
def multiply(b: Rational):Rational = {
return new Rational(numerator * b.numerator, denominator * b.denominator)
}
def multiply(b: BigInt):Rational = {
return new Rational(numerator * b, 1)
}
def divide(b: Rational):Rational = {
return new Rational(numerator * b.denominator, denominator * b.numerator)
}
def devide(b: BigInt):Rational = {
return new Rational(numerator / b, 1)
}
def compute_lcm(m: BigInt, n:BigInt):BigInt = {
var a = m
var b = n
while ( a != b) {
if (a<b ) a = a + m
else b = b + n
}
return a
}
def compute_gcd(a:BigInt, b:BigInt) : BigInt = {
var first_number:BigInt = 0
var second_number:BigInt = 0
```

```
if (a>b) {
first_number = a
second_number = b
} else {
first_number = b
second_number = a
}
var remainder:BigInt = 1
while (remainder != 0) {
remainder = first_number % second_number
first_number = second_number
second_number = remainder
}
return first_number
}
def gcd(b:Rational) : Rational = {
val x:BigInt = compute_gcd(numerator, b.numerator)
val y:BigInt = compute_lcm(denominator, b.denominator)
return new Rational(x, y)
}
def gcd(b:BigInt) : Rational = {
val z:Rational = new Rational(b, 1)
return gcd(y)
}
def printObject = println("numerator =" + numerator + " denominator=" + denominator)
}
object RationalMain {
def main(args: Array[String]):Unit = {
val r1 = new Rational(15,12)
println("For RationaL Number r1")
r1.printObject
val r2 = new Rational(6,8)
println("For RationaL Number r2")
r2.printObject
val sum_r1_r2 = r1.sum(r2)
println(" For r1 + r2: ")
sum_r1_r2.printObject
val subtract_r1_r2 = r1.subtract(r2)
println(" For r1 - r2: ")
sum_r1_r2.printObject
subtract_r1_r2.printObject
val multiply_r1_r2 = r1.multiply(r2)
println(" For r1 * r2: ")
```

```
multiply_r1_r2.printObject
val divide_r1_r2 = r1.divide(r2)
println( " For r1 / r2: ")
divide_r1_r2.printObject
val gcd_r1_r2 = r1.gcd(r2)
println (" gcd r1 and r2: ")
gcd_r1_r2.printObject
val r3 = new Rational(15)
println("For RationaL Number r3")
r3.printObject
val r4 = new Rational(6)
println("For RationaL Number r4")
r4.printObject
val sum_r3_r4 = r3.sum(r4)
println(" For r3 + r4: ")
sum_r3_r4.printObject
val subtract_r3_r4 = r3.subtract(r4)
println(" For r3 - r4: ")
sum_r3_r4.printObject
subtract_r3_r4.printObject
val multiply_r3_r4 = r3.multiply(r4)
println(" For r3 * r4: ")
multiply_r3_r4.printObject
val divide_r3_r4 = r3.divide(r4)
println(" For r3 / r4: ")
divide_r3_r4.printObject
val gcd_r3_r4 = r3.gcd(r4)
println(" gcd r3 and r4: ")
gcd_r3_r4.printObject
}
}
```



Applications Places System
acadgild@localhost:~/assignment_14.1
File Edit View Search Terminal Tabs Help
acadgild@localhost:~
acadgild@localhost:~/assignment_14.1
acadgild@locall
acadgild@localhost:~/...

```
[acadgild@localhost assignment_14.1]$
[acadgild@localhost assignment_14.1]$
[acadgild@localhost assignment_14.1]$
[acadgild@localhost assignment_14.1]$
[acadgild@localhost assignment_14.1]$ scalac RationalMain.scala
[acadgild@localhost assignment_14.1]$ scala RationalMain
For RationaL Number r1
numerator =15 denominator=12
For RationaL Number r2
numerator =6 denominator=8
 For r1 + r2:
numerator =192 denominator=96
 For r1 - r2:
numerator =192 denominator=96
numerator =48 denominator=96
 For r1 * r2:
numerator =90 denominator=96
 For r1 / r2:
numerator =120 denominator=72
 gcd r1 and r2:
numerator =3 denominator=24
For RationaL Number r3
numerator =15 denominator=1
For RationaL Number r4
numerator =6 denominator=1
 For r3 + r4:
numerator =21 denominator=1
 For r3 - r4:
numerator =21 denominator=1
numerator =9 denominator=1
 For r3 * r4:
numerator =90 denominator=1
 For r3 / r4:
numerator =15 denominator=6
 gcd r3 and r4:
numerator =3 denominator=1
[acadgild@localhost assignment_14.1]$ 
```

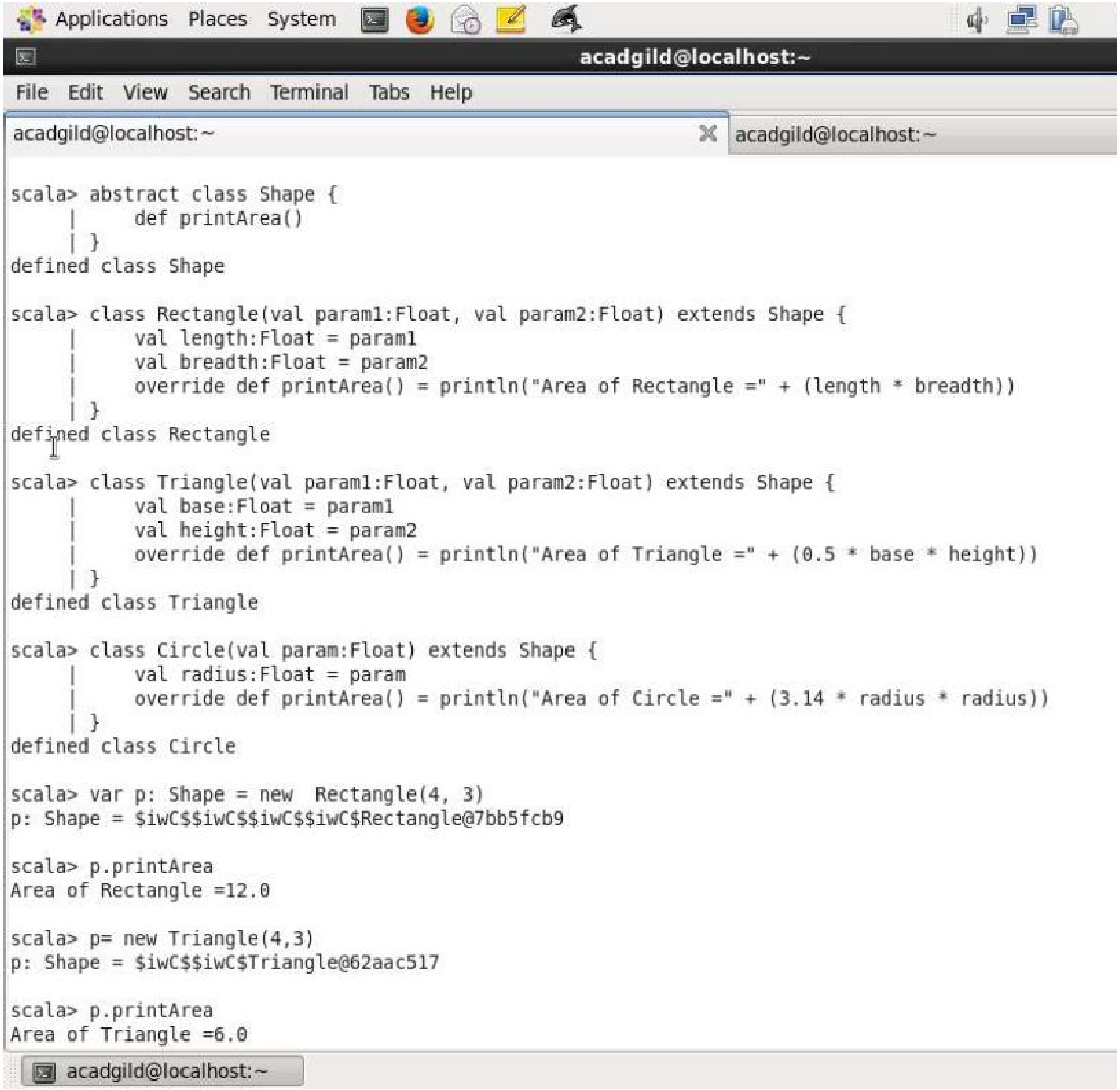
Task 3

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

```
abstract class Shape {
def printArea()
}
```

```
class Rectangle(val param1:Float, val param2:Float) extends Shape {
val length:Float = param1
val breadth:Float = param2
override def printArea() = println("Area of Rectangle =" + (length * breadth))
}


class Triangle(val param1:Float, val param2:Float) extends Shape {
val base:Float = param1
val height:Float = param2
override def printArea() = println("Area of Triangle =" + (0.5 * base * height))
}


class Circle(val param:Float) extends Shape {
val radius:Float = param
override def printArea() = println("Area of Circle =" + (3.14 * radius * radius))
}


p.printArea
p= new Triangle(4,3)
p.printArea
p= new Circle(4)
p.printArea
```

Screenshot:



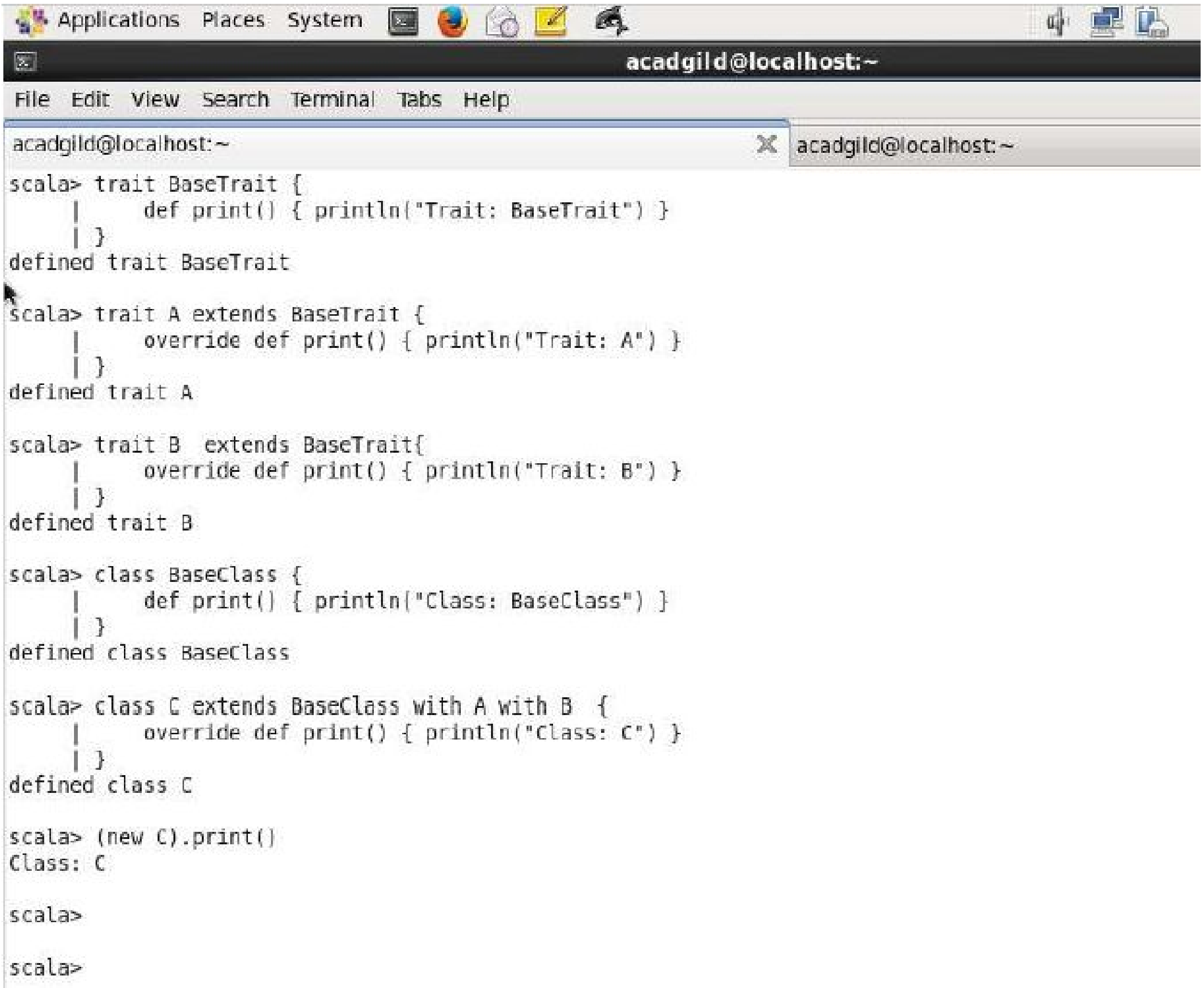
```
scala> abstract class Shape {
     |     def printArea()
     | }
defined class Shape

scala> class Rectangle(val param1:Float, val param2:Float) extends Shape {
     |     val length:Float = param1
     |     val breadth:Float = param2
     |     override def printArea() = println("Area of Rectangle =" + (length * breadth))
     | }
defined class Rectangle

scala> class Triangle(val param1:Float, val param2:Float) extends Shape {
     |     val base:Float = param1
     |     val height:Float = param2
     |     override def printArea() = println("Area of Triangle =" + (0.5 * base * height))
     | }
defined class Triangle

scala> class Circle(val param:Float) extends Shape {
     |     val radius:Float = param
     |     override def printArea() = println("Area of Circle =" + (3.14 * radius * radius))
     | }
defined class Circle

scala> var p: Shape = new  Rectangle(4, 3)
p: Shape = $iwC$$iwC$$iwC$$iwC$Rectangle@7bb5fcb9

scala> p.printArea
Area of Rectangle =12.0

scala> p= new Triangle(4,3)
p: Shape = $iwC$$iwC$Triangle@62aac517

scala> p.printArea
Area of Triangle =6.0
```

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

```
trait BaseTrait {
def print() { println("Trait: BaseTrait") }
}

trait A extends BaseTrait {
override def print() { println("Trait: A") }
}
```

```
trait B extends BaseTrait{
override def print() { println("Trait: B") }
}


class BaseClass {
def print() { println("Class: BaseClass") }
}


class C extends BaseClass with A with B {
override def print() { println("Class: C") }
}

(new C).print()
```

Screenshot:


Applications Places System
acadgild@localhost:~
File Edit View Search Terminal Tabs Help
acadgild@localhost:~
acadgild@localhost:~

```
scala> trait BaseTrait {
     |     def print() { println("Trait: BaseTrait") }
     | }
defined trait BaseTrait

scala> trait A extends BaseTrait {
     |     override def print() { println("Trait: A") }
     | }
defined trait A

scala> trait B  extends BaseTrait{
     |     override def print() { println("Trait: B") }
     | }
defined trait B

scala> class BaseClass {
     |     def print() { println("Class: BaseClass") }
     | }
defined class BaseClass

scala> class C extends BaseClass with A with B  {
     |     override def print() { println("Class: C") }
     | }
defined class C

scala> (new C).print()
Class: C

scala>

scala>
```

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.

```
def sum(a:Int, b:Int, c:Int) = a + b + c


def modifiedSum = sum(5, _:Int, _:Int)


def modifiedSquare(callback : (Int, Int) => Int, x:Int, y:Int):Int = {
val z = callback(x,y)
z * z
}


val p = modifiedSquare(modifiedSum, 7, 8)
println(p)


val q = modifiedSquare(modifiedSum, 3, 4)
println(q)
```

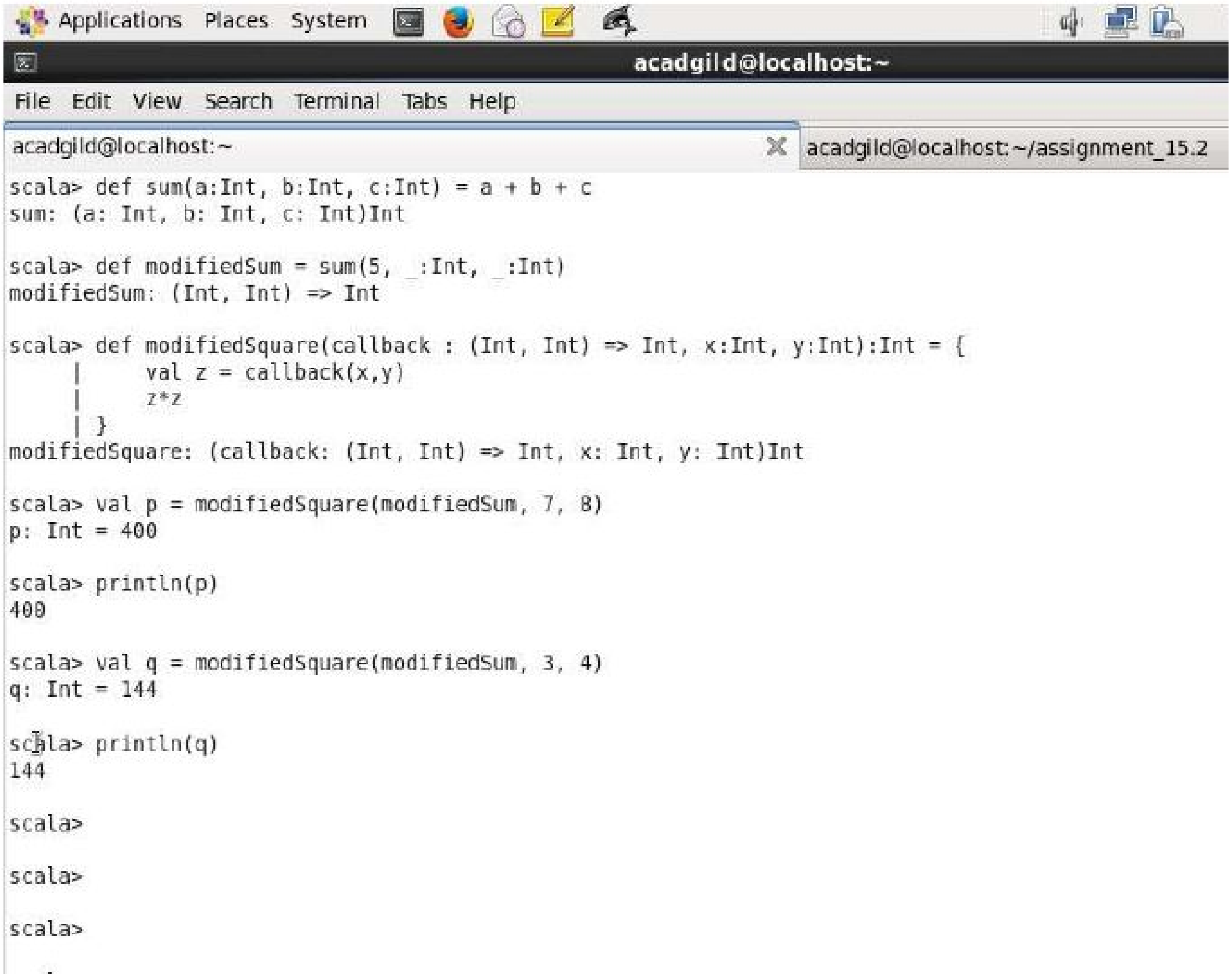
Screenshot is as below:

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

```
def findPrice(subject: String):Int = {
val price:Int = subject match {
case "Android" => 12999
case "Big Data Development" => 17999
case "Advanced Big Data Development" => 17999
case "Spark" => 19999
case _ => -1
}
return price
}
```

Step2: Call the method with various subjects than Android, Big Data Development, Advanced Big Data
Development, Spark, Java and return the corresponding price.
val p = findPrice("Android")
val p = findPrice("Big Data Development")
val p = findPrice("Advanced Big Data Development")
val p = findPrice("Spark")
val p = findPrice("Java")

Screenshot :

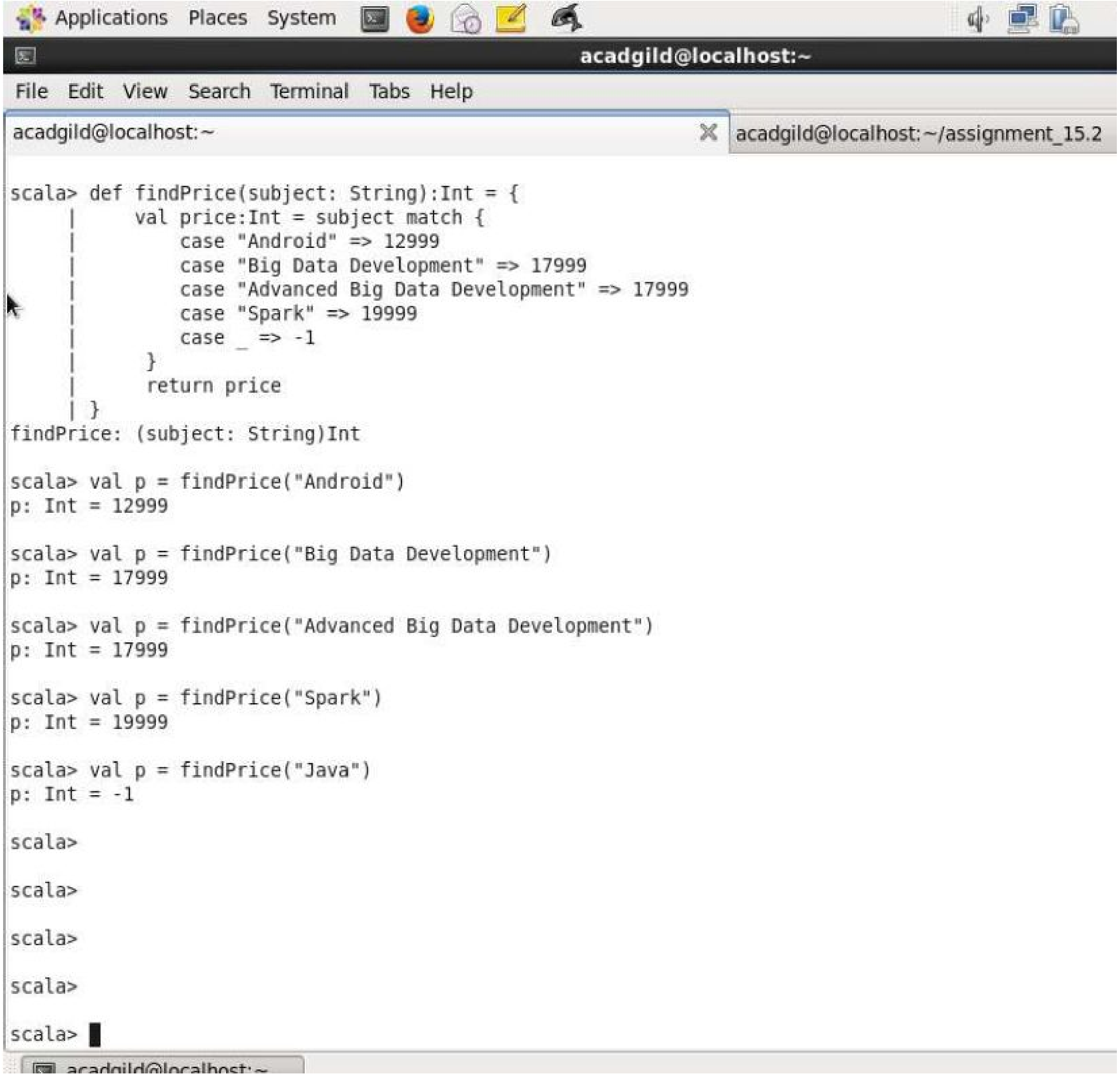


```
scala> def sum(a:Int, b:Int, c:Int) = a + b + c
sum: (a: Int, b: Int, c: Int)Int

scala> def modifiedSum = sum(5, _:Int, _:Int)
modifiedSum: (Int, Int) => Int

scala> def modifiedSquare(callback : (Int, Int) => Int, x:Int, y:Int):Int = {
     |     val z = callback(x,y)
     |     z*z
     | }
modifiedSquare: (callback: (Int, Int) => Int, x: Int, y: Int)Int

scala> val p = modifiedSquare(modifiedSum, 7, 8)
p: Int = 400

scala> println(p)
400

scala> val q = modifiedSquare(modifiedSum, 3, 4)
q: Int = 144

scala> println(q)
144

scala>

scala>

scala>
```

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

```
def findPrice(subject: String):Int = {
val price:Int = subject match {
case "Android" => 12999
case "Big Data Development" => 17999
case "Advanced Big Data Development" => 17999
case "Spark" => 19999
case _ => -1
}
return price
}


val p = findPrice("Android")
val p = findPrice("Big Data Development")
val p = findPrice("Advanced Big Data Development")
val p = findPrice("Spark")
val p = findPrice("Java")
```

Screenshot:


Applications Places System
acadgild@localhost:~
File Edit View Search Terminal Tabs Help
acadgild@localhost:~
acadgild@localhost:~/assignment_15.2
acadgild@localhost:~

```
scala> def findPrice(subject: String):Int = {
     |     val price:Int = subject match {
     |         case "Android" => 12999
     |         case "Big Data Development" => 17999
     |         case "Advanced Big Data Development" => 17999
     |         case "Spark" => 19999
     |         case _ => -1
     |      }
     |      return price
     | }
findPrice: (subject: String)Int

scala> val p = findPrice("Android")
p: Int = 12999

scala> val p = findPrice("Big Data Development")
p: Int = 17999

scala> val p = findPrice("Advanced Big Data Development")
p: Int = 17999

scala> val p = findPrice("Spark")
p: Int = 19999

scala> val p = findPrice("Java")
p: Int = -1

scala>

scala>

scala>

scala>

scala> 
```