

Task 1

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

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

Solution:

- Input N
- Initialize Fibonacci Sequence to blank
- Do while length of Fibonacci Sequence is less than N
- Generate Next Fibonacci Number
- Append Fibonacci Number to Fibonacci Sequence
- Return N th Number from the Fibonacci Sequence

Code is as below:

```
def find_fibonacci_sequence(N : Int) : String = {  
    var current_term:BigInt = 0  
    var next_term:BigInt = 1  
    var ret_value = ""  
    while (ret_value.length < N) {  
        ret_value += next_term.toString  
        var tmp = current_term  
        current_term = next_term  
        next_term += tmp  
    }  
    return ret_value.charAt(N - 1).toString  
}
```

Screenshot is as below.

```
File Edit View Search Terminal Help

scala>

scala> def find_fibonacci_sequence(N:Int):String = {
|   var current_term:BigInt = 0
|   var next_term:BigInt = 1
|   var ret_value = ""
|   while (ret_value.length < N) {
|       ret_value += next_term.toString
|       var tmp = current_term
|       current_term = next_term
|       next_term += tmp
|   }
|   return ret_value.charAt(N-1).toString
| }
find_fibonacci_sequence: (N: Int)String

scala> find_fibonacci_sequence(1)
res0: String = 1

scala> find_fibonacci_sequence(3)
res1: String = 2

scala> find_fibonacci_sequence(7)
res2: String = 1

scala> find_fibonacci_sequence(10)
res3: String = 1

scala> find_fibonacci_sequence(4)
res4: String = 3

scala> find_fibonacci_sequence(5)
res5: String = 5

scala> find_fibonacci_sequence(15)
res6: String = 8

scala> █
```

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

Solution:

- Input N
- Initialize Fibonacci Sequence to blank, current term to 0, next term to 1
- Use tail recursion to Generate Next Fibonacci Number assigning next term to current_term
- Append Fibonacci Number to Fibonacci Sequence, Assigning sum of current term, next_term to next_term
- If length of Fibonacci Sequence is greater than N then return Nth digit,
- Else apply recursion

Scala code is as below:

```
import scala.annotation.tailrec
```

```

def find_fibonacci_sequence(N : Int) : String = {

  @tailrec

  def find_fibonacci(N: Int, orig_N: Int, current_tem: BigInt = 0, next_tem: BigInt = 1, fib_seq:String = "") :
  String = {

    if (fib_seq.length >= orig_N) {

      return fib_seq.charAt(orig_N - 1).toString

    } else {

      var new_fib_seq = fib_seq + next_tem.toString

      return find_fibonacci(N-1, orig_N, next_tem, (current_tem + next_tem), fib_seq)

    }

  }

  return find_fibonacci(N, N)

}

```

```

File Edit View Search Terminal Help
scala> import scala.annotation.tailrec
import scala.annotation.tailrec

scala> def find_fibonacci_sequence(N: Int): String = {
  |   @tailrec
  |   def find_fibonacci(N: Int, orig_N: Int, current_term: BigInt = 0, next_term: BigInt = 1, fib_seq: String = ""): String = {
  |     if (fib_seq.length >= orig_N) {
  |       return fib_seq.charAt(orig_N - 1).toString
  |     } else {
  |       var new_fib_seq = fib_seq + next_term.toString
  |       return find_fibonacci(N-1, orig_N, next_term, current_term + next_term, new_fib_seq)
  |     }
  |   }
  |   find_fibonacci(N, N)
  | }
find_fibonacci_sequence: (N: Int)String

scala> find_fibonacci_sequence(1)
res0: String = 1

scala> find_fibonacci_sequence(3)
res1: String = 2

scala> find_fibonacci_sequence(7)
res2: String = 1

scala> find_fibonacci_sequence(10)
res3: String = 1

scala> find_fibonacci_sequence(4)
res4: String = 3

scala> find_fibonacci_sequence(5)
res5: String = 5

scala> find_fibonacci_sequence(15)
res6: String = 8

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)

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.

Solution:

I defined a class Rational and two private attributes numerator and denominator which are of type BigInt. I have chosen BigInt so that same program can be used for very big numbers. I defined one auxiliary constructor so that whole number can be used. Defined the following method:

sum: There are two overloaded methods, first one take Rational as a parameter and returns sum as a rational number. Second one takes BigInt as parameter and returns sum as a Rational number. Sum is computed by:

$$\frac{(\text{Numerator of Object} * \text{Denominator of parameter} + \text{Denominator of Object} * \text{Numerator of parameter})}{(\text{Denominator of Object} * \text{Denominator of Parameter})}$$

subtract: There are two overloaded methods, first one take Rational as a parameter and returns subtracts from the object and return as a rational number. Second one takes BigInt as parameter and subtracts the number from the object and returns as a Rational number . Subtract is computed by:

$$\frac{(\text{Numerator of Object} * \text{Denominator of parameter} - \text{Denominator of Object} * \text{Numerator of parameter})}{(\text{Denominator of Object} * \text{Denominator of Parameter})}$$

multiply: There are two overloaded methods, first one take Rational as a parameter and returns multiplies with the object and return as a rational number. Second one takes BigInt as parameter and multiply with the number from the object and returns as a Rational number. Multiply is computed by:

$$\frac{(\text{Numerator of Object} * \text{Numerator of Parameter})}{(\text{Denominator of Object} * \text{Denominator of Parameter})}$$

divide: There are two overloaded methods, first one take Rational as a parameter and divides from the object and return as a rational number. Second one takes BigInt as parameter and divides from the object and returns as a Rational number :

$$\frac{(\text{Numerator of Object} * \text{Denominator of Parameter})}{(\text{Denominator of Object} * \text{Numerator of Parameter})}$$

gcd: There are two overloaded methods, first one take Rational as a parameter and finds gcd with the object and return as a rational number. Second one takes BigInt as parameter and finds gcd with the object and returns as a Rational number. Gcd of Rational Number is calculated by:

GCD of numerator of Object and Parameter/ LCM of denominator of Object and Parameter

LCM is computed using a method compute_lcm which takes two BigInt numbers and computes the LCM, by repeatedly summing till both the numbers are equal

GCD is computed by a method compute_gcd, This method first finds greater of two numbers and assign to first_number and second_number respectively. Remainder is calculated by using modulus operator (%) between first_number and second_number. This steps is done repeated in while loop till remainder is 0. When remainder becomes 0, first_number is returned as the gcd.

I have also defined a method printObject which will print the numerator and denominator value of Rational object

I have written one singleton Object RationalMain having main method. Here I have taken two sets of Rational Number. In one set I have taken two Rational Number 15/12 and 6/8 and calculated sum, subtract, multiply, divide, gcd

In another set I have taken two whole Numbers 15 and 6 and computed sum, subtract, multiply, divide, gcd

I have created a file RationalMain.scala and its content is as below:

```
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
}
}

```

I have compiled the code using

scalac RationalMain.scala

And Run using:

scala RationalMain

The screenshot of compilation and output is as below:

```

acadgild@localhost:~/assignment_14.1
File Edit View Search Terminal Tabs Help
acadgild@localhost:~ acadgild@localhost:~/assignment_14.1 acadgild@localhost:~/assignment_13.1
[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]$

```

Screenshots of Source Code is as below:

The screenshot shows a terminal window titled "acadgild@localhost:~/assignment_14.1". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", "Tabs", and "Help". There are three tabs open: "acadgild@localhost:~", "acadgild@localhost:~/assignment_14.1" (which is the active tab), and "acadgild@localhost:~/assignment_13.1". The terminal displays the following Scala code:

```
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  
  }  
}
```

The bottom right of the terminal shows the line numbers "1,13" and a "Top" button. The status bar at the very bottom shows the prompt "acadgild@localhost:~/..." and some system icons.

Applications Places System Wed Dec 6, 11:10 AM acadgild

acadgild@localhost:~/assignment_14.1

File Edit View Search Terminal Tabs Help

acadgild@localhost:~ acadgild@localhost:~/assignment_14.1 acadgild@localhost:~/assignment_13.1

```
}  
  
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)  
}
```

72,0-1 42%

acadgild@localhost:~/...

Applications Places System Wed Dec 6, 11:11 AM acadgild

acadgild@localhost:~/assignment_14.1

File Edit View Search Terminal Tabs Help

acadgild@localhost:~ acadgild@localhost:~/assignment_14.1 acadgild@localhost:~/assignment_13.1

```
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
  }
}
```

108,13 85%

acadgild@localhost:~/...

Applications Places System Wed Dec 6, 11:12 AM acadgild

acadgild@localhost:~/assignment_14.1

File Edit View Search Terminal Tabs Help

acadgild@localhost:~ acadgild@localhost:~/assignment_14.1 acadgild@localhost:~/assignment_13.1

```
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
}
```

120,2 Bot

acadgild@localhost:~/...

```
acadmild@localhost:~  
File Edit View Search Terminal Tabs Help  
acadmild@localhost:~ acadmild@localhost:~/assignment_13.2  
scala> def fibonacciSeq(n : Int) : String = {  
  @tailrec  
  def fibonacci(n: Int, original_n: Int, term1: BigInt = 0, term2: BigInt = 1, x:String = "") : String = {  
    if (x.length >= original_n) {  
      return x.charAt(original_n - 1).toString  
    } else {  
      var y = x + term2.toString  
      return fibonacci(n-1, original_n, term2, (term1 + term2), y)  
    }  
  }  
  return fibonacci(n, n)  
}  
fibonacciSeq: (n: Int)String  
scala> fibonacciSeq(1)  
res53: String = 1  
scala> fibonacciSeq(2)  
res54: String = 1  
scala> fibonacciSeq(3)  
res55: String = 2  
scala> fibonacciSeq(4)  
res56: String = 3  
scala> fibonacciSeq(5)  
res57: String = 5  
scala> fibonacciSeq(6)  
res58: String = 8  
scala> fibonacciSeq(7)  
res59: String = 1  
scala> 
```

Task 3

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

Solution:

Step1: I defined a abstract class Shape which has one method printArea.

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

Step2: Class Rectangle extends from Shape having attributes length and breadth and overrides the method printArea. Area is calculated using length*breadth

```

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))
}

```

Step3: Class Triangle extends from Shape having attributes base and height and overrides the method printArea. area is calculated by $0.5 * \text{base} * \text{height}$

```

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 extends from Shape having attributes radius and overrides method printArea. area is calculated by $3.14 * \text{radius} * \text{radius}$

```

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

```

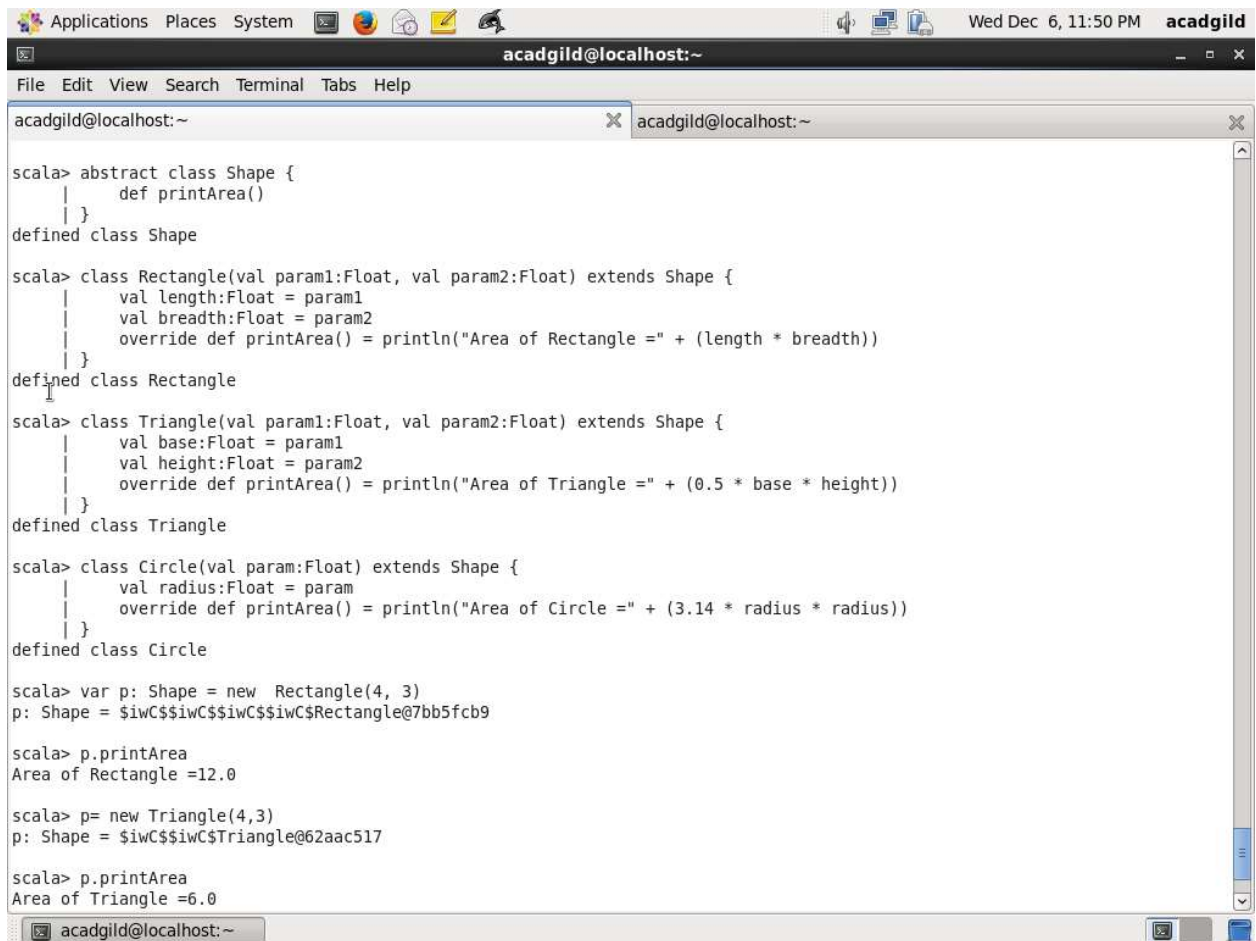
Next, I called created object Rectangle, Triangle and Circle and call its printArea method

```

var p: Shape = new Rectangle(4, 3)
p.printArea
p= new Triangle(4,3)
p.printArea
p= new Circle(4)
p.printArea

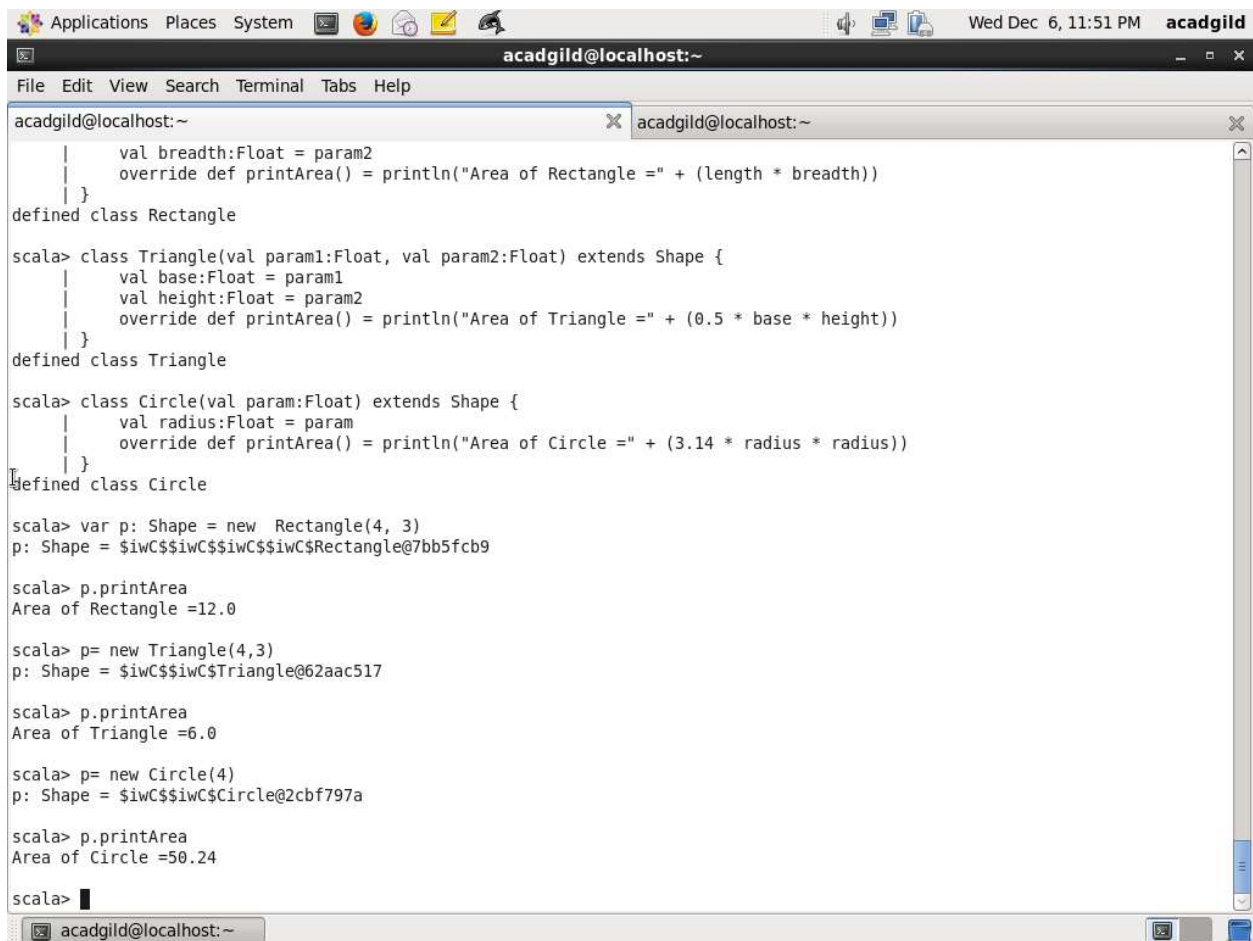
```

Screenshot is as below:



The screenshot shows a terminal window titled 'acadgild@localhost:~' with a menu bar (File, Edit, View, Search, Terminal, Tabs, Help). The terminal contains the following Scala code and output:

```
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
```



```
acadgild@localhost:~  
File Edit View Search Terminal Tabs Help  
acadgild@localhost:~ acadgild@localhost:~  
|    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$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  
  
scala> p= new Circle(4)  
p: Shape = $iwC$$iwC$Circle@2cbf797a  
  
scala> p.printArea  
Area of Circle =50.24  
  
scala> █
```

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

Solution:

Multiple inheritance can be achieved using trait.

Step1: I defined BaseTrait which has a method print.

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

Step2:

Traits A and B extends from BaseTrait and override method print:

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

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

Step3: Defined a class BaseClass which has method print

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

Step4: Defined a class C which extends from class BaseClass and also extends traits A and B and overrides method print:

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

Step5; Create a object of C and call print method

```
(new C).print()
```

Screenshot is as below:

```
acadgild@localhost:~  
File Edit View Search Terminal Tabs Help  
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>  
scala>  
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.

Solution:

Step1: Define method sum which take three integer arguments and sum them

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

Step2: Define partial function modifiedSum which refines sum function by making first argument constant value 5 and takes two arguments

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

Step2: Define function modifiedSquare which take a method callback as first parameter and two more integer parameters. It calls callback methd two two integer aguments and square the result

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

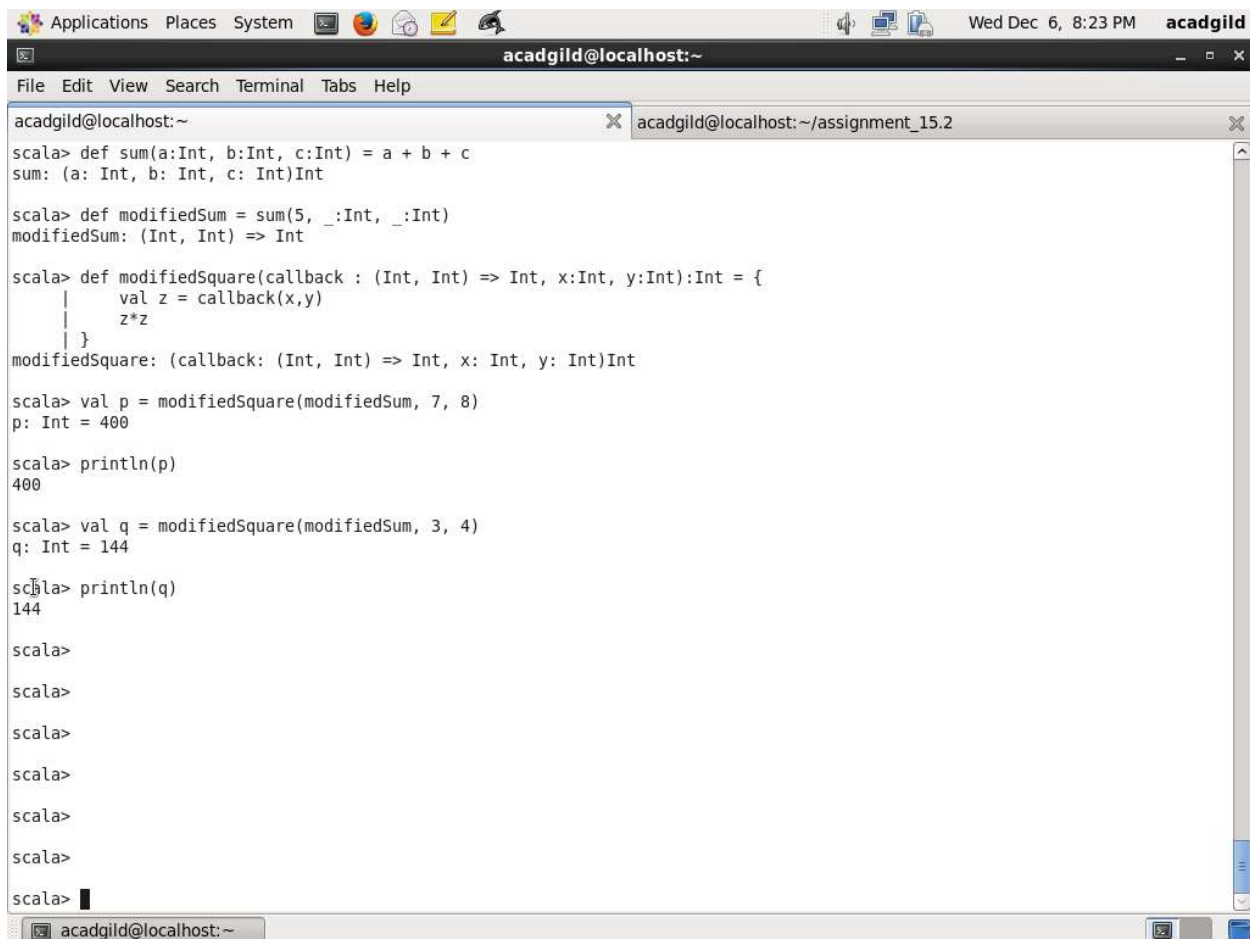
Step3: Call modifiedSqaure with modifiedSum as callback method and arguments 7 and 8

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

Step3: Call modifiedSqaure with modifiedSum as callback method and arguments 3 and 4

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

Screenshot is as below:



```
acadgild@localhost:~  
File Edit View Search Terminal Tabs Help  
acadgild@localhost:~ acadgild@localhost:~/assignment_15.2  
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>  
scala>  
scala>  
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

Solution:

Note: In the question, there are two same subject Big Data Development gives, if I try I am getting warning. So modified second one as Advanced Big Data Development

Step1: Define function findPrice which will take subject as input and will return price of subject





```
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 is as below:

Applications Places System     Wed Dec 6, 9:37 PM acadgild

acadgild@localhost:~

File Edit View Search Terminal Tabs Help

acadgild@localhost:~ acadgild@localhost:~/assignment_15.2

```
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>
```

acadgild@localhost:~