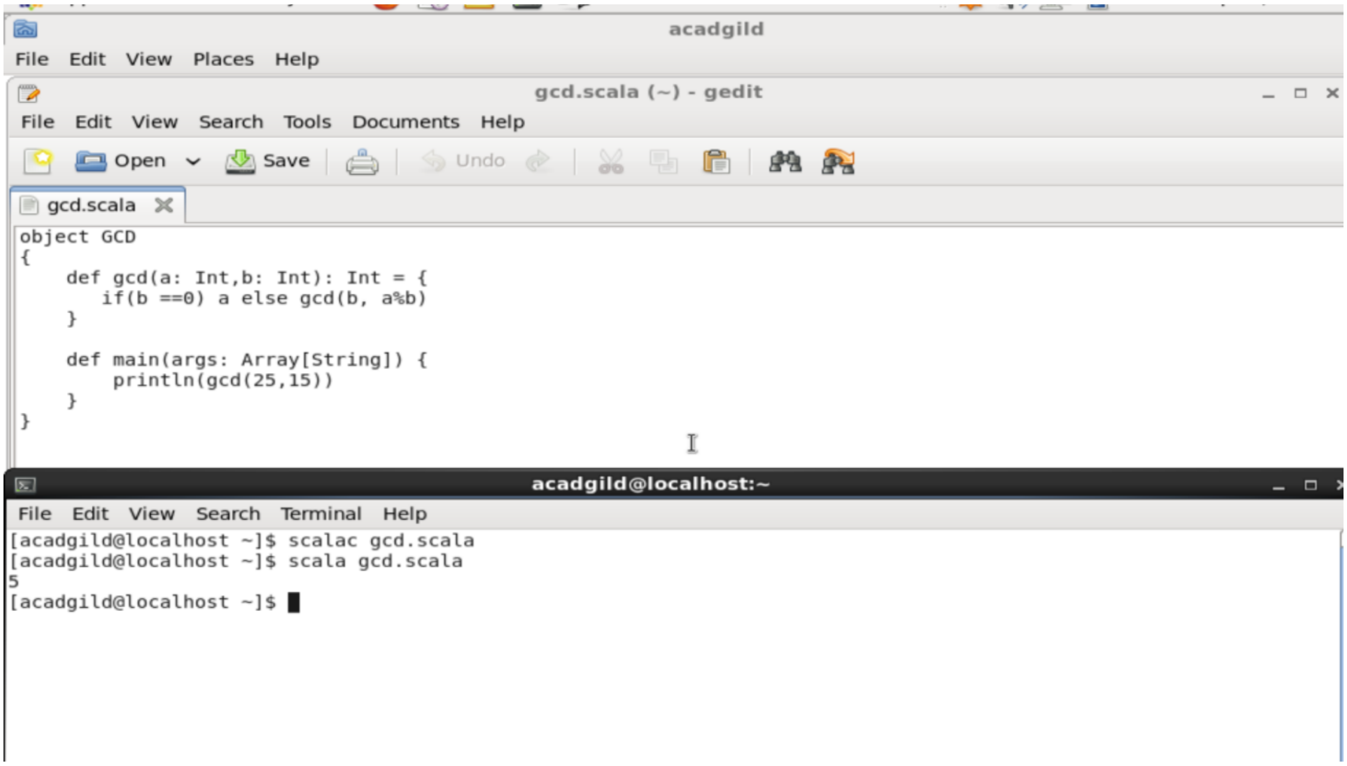
Session 15: SCALA BASICS 2

Assignment 1

Task 1

Create a scala application to find the GCD of two numbers.


acadgild
File Edit View Places Help
gcd.scala (~) - gedit
File Edit View Search Tools Documents Help
Open
Save
Undo
gcd.scala
acadgild@localhost:~
File Edit View Search Terminal Help

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

    def main(args: Array[String]) {
        println(gcd(25,15))
    }
}
```

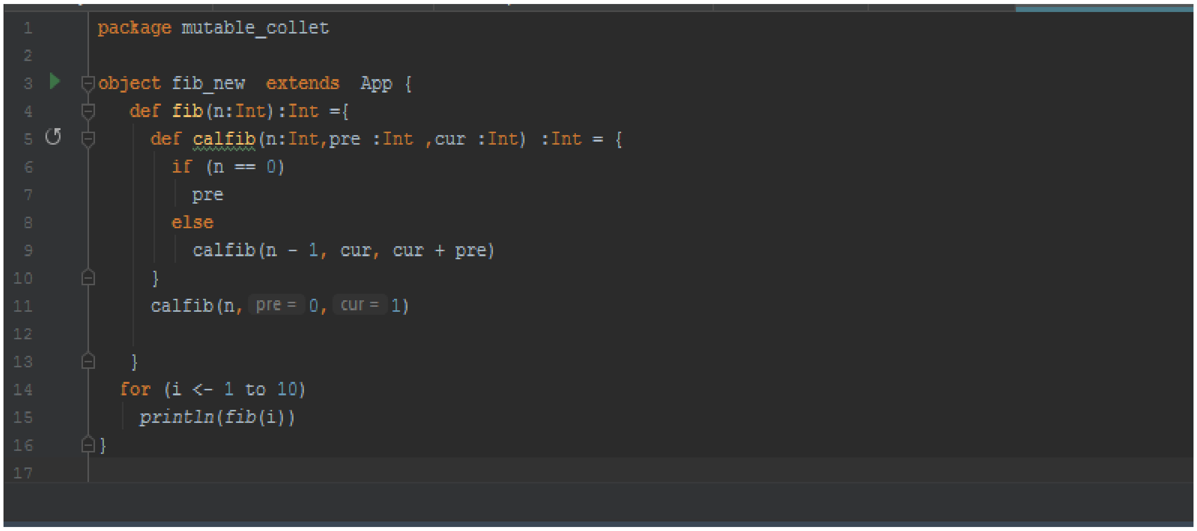
```
[acadgild@localhost ~]$ scalac gcd.scala
[acadgild@localhost ~]$ scala gcd.scala
5
[acadgild@localhost ~]$ 
```

Task 2

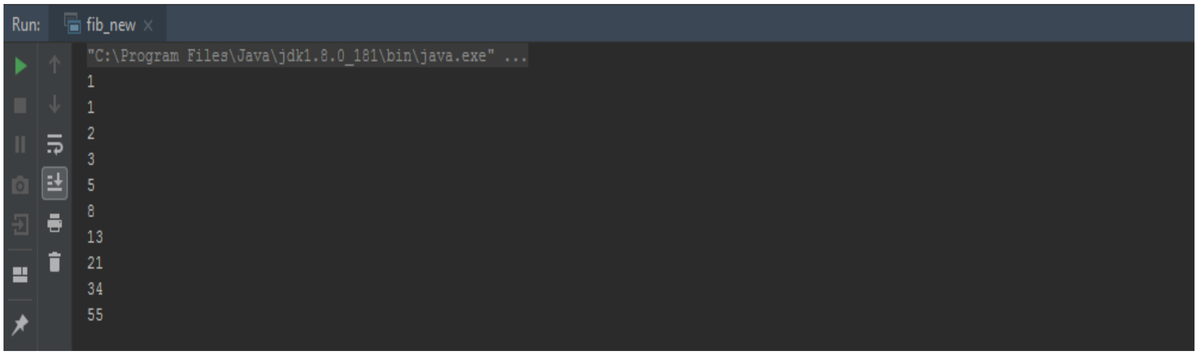
Fibonacci series (starting from 1) written in order without any spaces in between, thus producing a sequence of digits.

Write a Scala application to find the Nth digit in the sequence.

- Write the function using standard for loop

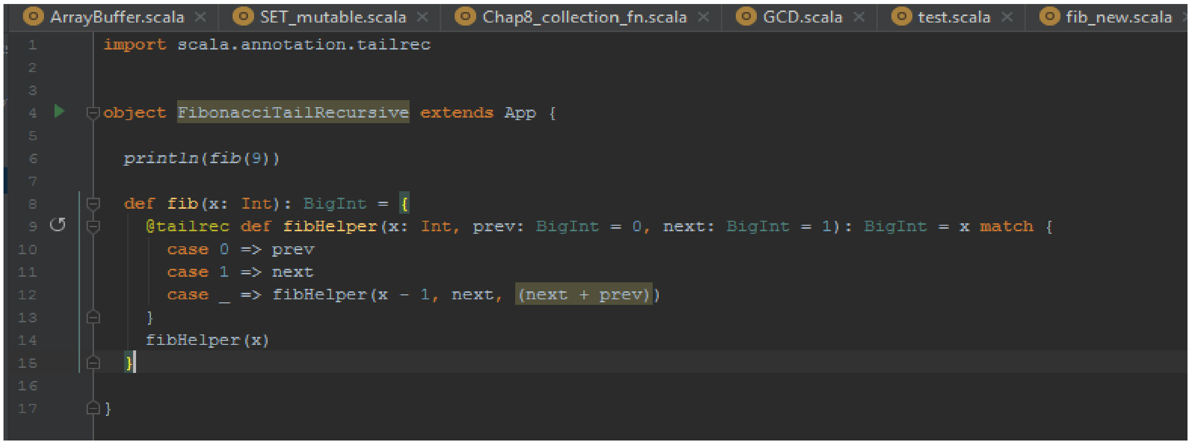


```scala
package mutable_collet

object fib_new  extends  App {
   def fib(n:Int):Int ={
     def calfib(n:Int,pre :Int ,cur :Int)  :Int = {
       if  (n == 0)
         pre
       else
         calfib(n - 1, cur,  cur + pre)
     }
     calfib(n, pre = 0, cur = 1)

   }
  for (i <- 1 to 10)
    println(fib(i))
}
```

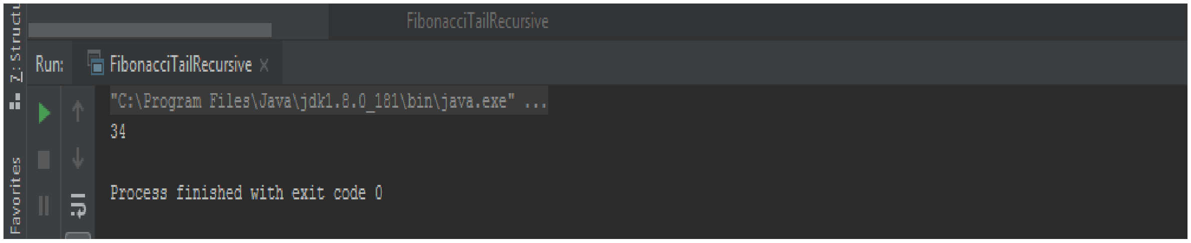


```
"C:\Program Files\Java\jdk1.8.0_181\bin\java.exe" ...
1
1
2
3
5
8
13
21
34
55
```

➢ Write the function using recursion



```scala
import scala.annotation.tailrec


object FibonacciTailRecursive extends App {

  println(fib(9))

  def fib(x: Int): BigInt = {
    @tailrec def fibHelper(x: Int, prev: BigInt = 0, next: BigInt = 1): BigInt = x match {
      case 0 => prev
      case 1 => next
      case _ => fibHelper(x - 1, next, (next + prev))
    }
    fibHelper(x)
  }

}
```

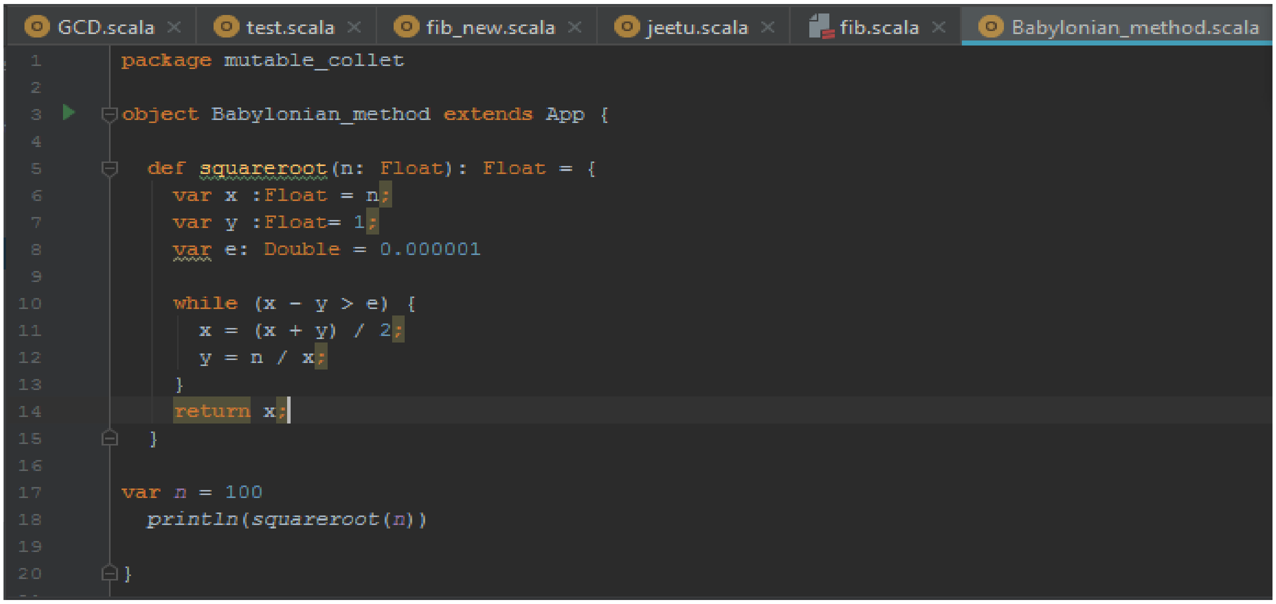


```
"C:\Program Files\Java\jdk1.8.0_181\bin\java.exe" ...
34

Process finished with exit code 0
```

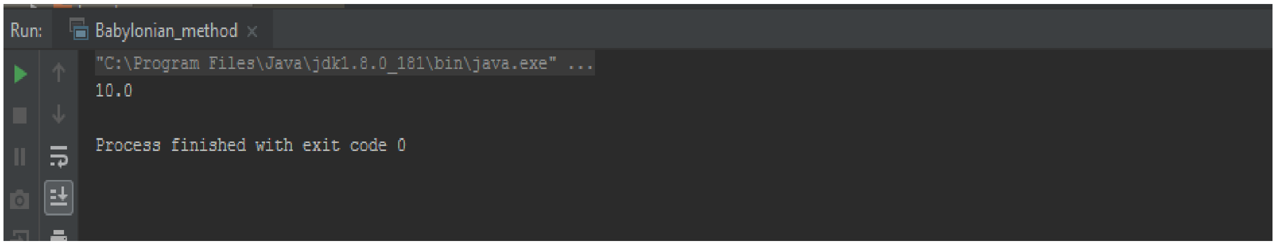
Task 3

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 the following until desired approximation is achieved.
   a) Get the next approximation for root using average of x and y
   b) Set y=n/x



```
package mutable_collet

object Babylonian_method extends App {

  def squareroot(n: Float): Float = {
    var x :Float = n;
    var y :Float= 1;
    var e: Double = 0.000001

    while (x - y > e) {
      x = (x + y) / 2;
      y = n / x;
    }
    return x;
  }

var n = 100
  println(squareroot(n))

}
```



```
"C:\Program Files\Java\jdk1.8.0_181\bin\java.exe" ...
10.0

Process finished with exit code 0
```