Bigdata Assignment 5

Task 1

Given a list of strings - List[String] (“alpha”, “gamma”, “omega”, “zeta”, “beta”)

- Find count of all strings with length 4.
- Convert the list of string to a list of integers, where each string is mapped to its corresponding length.
- Find count of all strings which contain alphabet ‘m’.
- Find the count of all strings which start with the alphabet ‘a’.

Code:

```
val list=List[String]("alpha","beta","gamma","omega","zeta")
```

Output:

```
scala> println(list.count(z=>z.length==4))
2
```

Code:

```
def toMatch(arg:String)=arg match{
   | case "_m_"=>print(c+1)
   | case _ =>print("Match not found")
   | }
```

Task 2

Create a list of tuples, where the 1st element of the tuple is an int and the second element is a string.
Example - ((1, ‘alpha’), (2, ‘beta’), (3, ‘gamma’), (4, ‘zeta’), (5, ‘omega’))
- For the above list, print the numbers where the corresponding string length is 4.
- find the average of all numbers, where the corresponding string contains alphabet ‘m’ or alphabet ‘z’.

Code:

```
val tuple=List[Any]((1,"alpha"),(2,"beta"),(3,"gamma"),(4,"zeta"),(5,"omega"))
```

Task 3

Create a Scala application to find the GCD of two numbers

Code:

Bigdata Assignment 5

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

Output:

```
scala>  def gcd(a: Int,b: Int): Int = {
      |
      |         if(b ==0) a else gcd(b, a%b)
      |
      |      }
gcd: (a: Int, b: Int)Int
```

```
scala> println(gcd(27,45))
9
```

<u>Task 4</u>

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

Code:

```
def fib1( n : Int) : Int = n match {
  case 0 | 1 => n
  case _ => fib1( n-1 ) + fib1( n-2 )
}

def fib2( n : Int ) : Int = {
 var a = 0
 var b = 1
 var i = 0

 while( i < n ) {
  val c = a + b
  a = b
  b = c
  i = i + 1
 }
 return a
```

Bigdata Assignment 5

```
}
def fibSeq(n: Int): List[Int] = {
  var ret = scala.collection.mutable.ListBuffer[Int](0, 1)
  while (ret(ret.length - 1) < n) {
    val temp = ret(ret.length - 1) + ret(ret.length - 2)
    if (temp >= n) {
      return ret.toList
    }
    ret += temp
  }
  ret.toList
 }
def nthFib(n: Int): Int = {
 var x = 0
 var y = 1
 for (_ <- 1 until n) {
  val temp = x + y
  x = y
  y = temp
 }
 y
}
def fib_rec(n:Long):Long = {
 def fib_recursion(n:Long, a:Long, b:Long):Long = {
  if(n == 0) a
  else fib_recursion(n - 1, b, a + b)
 }
 return fib_recursion(n, 0, 1)
}

println("fibonacci series of = "+ fib1(10))
println("fibonacci series of ="+fib2(20))
println("list if fib series="+fibSeq(15))
println("nth fibonacci value using for loop = "+nthFib(7))
println("nth fibonacci value using recursion function ="+fib_rec(18))
```

Output:

Attached the images at the last

<u>Task 5</u>

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

Bigdata Assignment 5

Code:

```
def square(n:Double):Stream[Double]=
   | {
   |   def square(guess:Double,n:Double):Stream[Double]={
   |         Stream.cons(guess,square(0.5*(guess+n/guess),n))
   |         }
   |   square(1,n)
   | }
```

Output:

```
scala>  def square(n:Double):Stream[Double]=
     | {
     |   def square(guess:Double,n:Double):Stream[Double]={
     |           Stream.cons(guess,square(0.5*(guess+n/guess),n))
     |           }
     |   square(1,n)
     | }
square: (n: Double)Stream[Double]

scala> square(9)
res13: Stream[Double] = Stream(1.0, ?)

scala> val iterations = 5
iterations: Int = 5

scala> square(9)(iterations - 1)
res14: Double = 3.00009155413138
```

Task 3 Output:

Bigdata Assignment 5

```
scala> def fib1( n : Int) : Int = n match {
     |    case 0 | 1 => n
     |    case _ => fib1( n-1 ) + fib1( n-2 )
     | }
fib1: (n: Int)Int

scala> def fib2( n : Int ) : Int = {
     |   var a = 0
     |   var b = 1
     |   var i = 0
     |
     |   while( i < n ) {
     |     val c = a + b
     |     a = b
     |     b = c
     |     i = i + 1
     |   }
     |   return a
     | }
fib2: (n: Int)Int

scala> def fibSeq(n: Int): List[Int] = {
     |     var ret = scala.collection.mutable.ListBuffer[Int](0, 1)
     |     while (ret(ret.length - 1) < n) {
     |       val temp = ret(ret.length - 1) + ret(ret.length - 2)
     |       if (temp >= n) {
     |         return ret.toList
     |       }
     |       ret += temp
     |     }
     |     ret.toList
     |   }
fibSeq: (n: Int)List[Int]
```

Bigdata Assignment 5

```
scala> def nthFib(n: Int): Int = {
     |   var x = 0
     |   var y = 1
     |   for (_ <- 1 until n) {
     |     val temp = x + y
     |     x = y
     |     y = temp
     |   }
     |   y
     | }
nthFib: (n: Int)Int

scala> def fib_rec(n:Long):Long = {
     |   def fib_recursion(n:Long, a:Long, b:Long):Long = {
     |     if(n == 0) a
     |     else fib_recursion(n - 1, b, a + b)
     |   }
     |   return fib_recursion(n, 0, 1)
     | }
fib_rec: (n: Long)Long

scala> println("fibonacci series of = "+ fib1(10))
fibonacci series of = 55

scala> println("fibonacci series of ="+fib2(20))
fibonacci series of =6765

scala> println("list if fib series="+fibSeq(15))
list if fib series=List(0, 1, 1, 2, 3, 5, 8, 13)

scala> println("nth fibonacci value using for loop = "+nthFib(7))
nth fibonacci value using for loop = 13

scala> println("nth fibonacci value using recursion function ="+fib_rec(18))
nth fibonacci value using recursion function =2584

scala>
```