```scala
package functions

import scala.collection.mutable
import scala.util.Random

def nextLetter(rand: Random): Char = {
  val randInt = rand.nextInt(52)
  if (randInt <= 25) {
    ('A' + randInt).toChar
  } else
    ('a' + (randInt - 26)).toChar
}

def randSet(rand: Random, count: Int, minIncl: Int, maxExcl: Int): Set[Int] = {
  if (count < 0 || minIncl > maxExcl || (minIncl == maxExcl && count > 0)) {
    throw IllegalArgumentException()
  }

  var ret = Set[Int]()
  val numRange = maxExcl - minIncl

  if (numRange - count > 5000) {
    while ret.size < count do ret = ret + (rand.between(minIncl, maxExcl))
    ret
  } else {
    ret = (minIncl until (maxExcl - 1)).toSet
    while ret.size > count do ret = ret - ret.toSeq(rand.between(0, ret.size))
    ret
  }
}
def makePassword(rand: Random, len: Int, specials: IndexedSeq[Char], specialCount: Int): String = {
  if (specialCount > len || (specials.isEmpty && specialCount > 0)) {
    throw IllegalArgumentException()
  }
  val sb = StringBuilder()
  val specialInd = randSet(rand, specialCount, 0, len)
  for i <- 0 until len do {
    if specialInd.contains(i) then sb.addOne(specials(i % specials.size))
    else sb.addOne(nextLetter(rand))
  }
  sb.result()
}

def findPosition[A](iterator: Iterator[A], target: A): Long = {
  var ret = 1
  while iterator.next() != target do {
    ret += 1
  }
  ret
}
def findDoublet[A](iterator: Iterator[A]): (A, A) = {
  if (!iterator.hasNext) {
    throw IllegalArgumentException()
  }
  var prev = iterator.next()
  while (iterator.hasNext) {
    val curr = iterator.next()
    if (curr == prev) {
      return (prev,curr)
    }
    prev = curr
  }
  throw IllegalArgumentException("No Duplicates Found")
}

def toCamelCase(str: String): String   = {
  val sb = StringBuilder()
  var nextUpper = false
  for char <- str do {
    if (char == '_') {
      nextUpper = true
    } else {
      sb.addOne(if (nextUpper) char.toUpper else char)
      nextUpper = false
    }
  }
  sb.result()
}
def fromCamelCase(str: String): String = {
  val sb = StringBuilder()
  for char <- str do {
    if (char.isUpper) {
      sb.addOne('_')
      sb.addOne(char.toLower)
    } else {
      sb.addOne(char)
    }
  }
  sb.result()
}

def counts[A](values: A*): Map[A, Int] = {
  var map = Map[A, Int]()
  for (num <- values) {
    if (map.contains(num)) {
      map = map.updated(num, (map(num) + 1))
    } else {
      map = map.updated(num, 1)
    }
  }
  map
}
def expand[A](counts: Map[A, Int]): Seq[A] = {
  var ret = Seq[A]()
  for (key <- counts.keys) {
    ret = ret.concat(Seq.fill(counts(key))(key))
  }
  ret
}

def mostFrequent[A](first: A, more: A*): A = {
  val map = counts(first +: more*)
  var most = 0
  var ret = first
  for (key <- map.keys) {
    if (map(key) > most) {
      ret = key
      most = map(key)
    }
  }
  ret
}

def buffon(needles: Iterator[(Float, Float)]): Double = {

  var crosses = 0.0
  var num_needles = 0
  for ((x,y) <- needles) {
    if (x == 0) {
      crosses += 1
    }
    val opp = Math.sin(y)
    if (opp + x >= 1) {
      crosses += 1
    }
    num_needles += 1
  }
  val p = crosses / num_needles
  2/p
}

// Bonus question. Leave as is if bonus is not implemented.
```

```
def makeMagicSquare(n: Int): Array[Array[Int
]]         = ???
def printMagicSquare(array: Array[Array[Int]]): String
 = ???

```