```
object hoja1_2020 {
  println("Welcome to the Scala worksheet")       //> Welcome to the Scala
worksheet

   import util.Random

  def poner (col: Int, pos:Int, l:List[Int]): List[Int]= {
      if (l.isEmpty) Nil
      else if (pos==1) col::l.tail
      else l.head::poner (col, (pos-1), l.tail) }
                                                  //> poner: (col: Int, pos:
Int, l: List[Int])List[Int]


val listi = List(0,0,0,0,0,0,0,0,0,0)             //> listi  : List[Int] =
List(0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
val liston = List(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
                                                  //> liston  : List[Int] =
List(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                                  //| , 0, 0, 0, 0, 0, 0, 0,
0, 0, 0)
val t1:List[Int] = poner (7, 2, liston)           //> t1  : List[Int] =
List(0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                                  //| 0, 0, 0, 0, 0, 0, 0,
0, 0)


val t2:List[Int] =poner (7, 3, t1)                //> t2  : List[Int] =
List(0, 7, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                                  //| 0, 0, 0, 0, 0, 0, 0,
0, 0)




def insertar_aux (color: Int, pos: Int, tablero: List[Int]): List[Int] = {
           if (pos==1) color::tablero.tail
      else tablero.head::insertar_aux(color, pos-1, tablero.tail)
      }                                           //> insertar_aux: (color:
Int, pos: Int, tablero: List[Int])List[Int]

val t3:List[Int]= insertar_aux (7,4,listi)        //> t3  : List[Int] =
List(0, 0, 0, 7, 0, 0, 0, 0, 0, 0)




def poner1 (col: Int, l:List[Int]): List[Int]= {
val pos= (1+Random.nextInt(27))
val l=List(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
if (l.isEmpty) Nil
else if (pos==1) col::l.tail
else l.head::poner1 (col, l.tail) }               //> poner1: (col: Int, l:
List[Int])List[Int]

val t11:List[Int] = poner1 (7, liston)            //> t11  : List[Int] =
List(0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                                  //| , 0, 0, 0, 0, 0, 0, 0,
0, 0, 0)
```

```
  def crearRandomCol ():Int={
  val random = util.Random
  val colo = random.nextInt(9)+1
  colo

  }                                                  //> crearRandomCol: ()Int


def ponerColo ( lis1:List[Int]): List[Int]= {
val pos5= crearRandom ()
val colo=crearRandomCol ()
println(pos5)
println(colo)

if (lis1.isEmpty) Nil
else if (pos5==1) colo::lis1.tail
else lis1.head::ponerColo ( lis1.tail) }          //> ponerColo: (lis1:
List[Int])List[Int]

val tfc:List[Int] =ponerColo ( lis1)              //> 1
                                                  //| 8
                                                  //| tfc  : List[Int] =
List(8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                                  //| 0, 0, 0, 0, 0, 0, 0, 0,
0, 0)
```

```
//Tratamiento de errores con throw

def imprimir (xs:List[(Int)]):Unit = xs match {
case Nil => println("---------")
case a1::a2::a3::a4::xs => println ("---------")
                                                    print("|")
                                                    print (a1)
                                                    print("|")
                                                    print (a2)
                                                    print ("|")
                                                    print (a3)
                                                    print ("|")
                                                    print (a4)
                                                    println ("|")
                                                    imprimir (xs)
case _ => throw new Error("TABLA ERRONEA")}        //> imprimir: (xs:
List[Int])Unit

imprimir(lis)                                       //> ---------
                                                    //| |15|46|44|92|
                                                    //| ---------
                                                    //| |94|50|60|29|
                                                    //| ---------
                                                    //| |24|36|47|90|
                                                    //| ---------
                                                    //| |3|49|74|53|
                                                    //| ---------
```

```
  def productElement(i: Int): Any = i match {
    case 0 => nombre
    case 1 => edad
    case _ => throw new IndexOutOfBoundsException(i.toString)
```

```
//     realiza una funcion que devuelva el número de divisores de N
excluyendo el 1 y el N.
//Primero definimos divisores_aux n x que devuelve el número de divisores de
n desde 2 a x

def divisores_aux (n:Int)(x:Int) :Int = x match {
case 1 => 0
case x => if (n % x == 0)  (1 + divisores_aux (n) (x-1))
          else divisores_aux (n) (x-1)}           //> divisores_aux: (n:
Int)(x: Int)Int


def divisores (n:Int): Int = divisores_aux (n) (n-1)
                                                  //> divisores: (n: Int)Int


divisores (10)                                    //> res5: Int = 2


divisores (100)                                   //> res6: Int = 7
```

```
def checkSign(x:Int):String ={
x match {
case a if a<0 => s"$a is a negative number"
case b if b>0 => s"$b is a positive number"
case c => s"$c neither positive nor negative"
}
}
val dd=checkSign(10)

checkSign: (x: Int)String
dd: String = 10 is a positive number



}
```