Exercise 2

a)

```
def func2(a: (Int, Int), b: (Int, Int)) = (a._1+a._2,
b._1*b._2)
```

b)

```
def maior(x:Int,y:Int) = if (x > y) x else y
def menor(x:Int,y:Int) = if (x > y) y else x
def maiorde3(x:Int, y:Int, z:Int) = (maior((maior(x,y)), z))
def middle3(x:Int, y:Int, z:Int) = (menor((maior(x,y)),
maior(menor(x,y),z)))
def largest(x: Int, y:Int, z: Int) = (maiorde3(x, y, z),
middle3(x,y,z))
```

c)

```
def menorde3(x:Int, y:Int, z:Int) = (menor((menor (x,y)), z))
def ordering(x:Int, y:Int, z:Int) = (largest(x, y, z)._1,
largest(x, y, z)._2, menorde3(x, y, z))
```

ou mais eficiente

```
def ordering1(x:Int, y:Int, z:Int) =
{
  val res = largest(x, y, z)(res._1, res._2, menorde3(x, y,
z))
}
```

d)

```
def constraint(x: Double, y: Double, z : Double) = x + y > z
&& x + z > y && y + z > x
```

e)

```
def abrev(str: String) = str.split(" ").head ++ " "++
str.split(" ").last
```

ou mais eficiente

```
def abrev1(str: String) =
{
  val lst: Array[String] = str.split(" "); lst.head ++ " " ++
lst.last
}
```

Exercise 3

a)

```
def exp(x: Int, y:Int): Int = if(y == 0) 1 else x * exp(x, y-
1)
```

b)

```
def pair(lst: List[Int]) = (lst.head, lst.last)
```

c)

```
def pairLength(lst: List[Int]) = (lst, lst.length)
```

d)

```
def sum(lst: List[Double]): Double = if(lst.isEmpty) 0 else
lst.head + sum(lst.tail)
def average(lst: List[Double]) = sum(lst) / lst.length
```

ou

```
def average1(lst: List[Double]) = lst.sum / lst.length
```

No caso de receber uma lista de Int e querer devolver um Double, p.e. :

```
def average2(lst: List[Int]) = lst.sum.toDouble / lst.length
```