Nome: Matheus Simões Minguini
SC3004023

Exercício 1

```
fun main(args: Array<String>) {

  val listaCursos: MutableList<String> = mutableListOf(
     "Sistemas para Dispositivos Móveis",
     "Análise e Desevolvimento de Sistemas")


  listaCursos.add("Técnico em Informática para Internet")
  listaCursos.add("Manutenção de Aeronaves")
  listaCursos.add("Técnico em Células")
  listaCursos.add("Processos Gerenciais")

  val listaCursosFiltrada: List<String> =
     listaCursos.filter { item -> item.contains("Sistemas")}

  listaCursosFiltrada.forEach { System.out.print(it + "\n") }
}
```

Exercício 2

```
fun main(args: Array<String>) {

  val familiaMap: MutableMap<String, String> =
     mutableMapOf(Pair("Pai", "Pedro"),
        Pair("Mãe", "Marcela"),
        Pair("Filho", "João"),
        Pair("Caçula", "Cadu"),
        Pair("Pet", "Paçoca"))

  familiaMap.forEach {(key, value) -> println("$key : $value")}
}
```

Exercício 3

```
fun soma(i: Int, j: Int) = i + j
fun cat(i: String, j: String): String = "${i}${j}"

fun main() {
  println(foo(10, 20, ::soma))
  println(bar("Ped", "ro", ::cat))
}

fun foo (primeiro_numero: Int, segundo_numero: Int, soma: (x: Int, y: Int) -> Int) : Int {
  return soma(primeiro_numero, segundo_numero)
}
```

```
fun bar(nome1: String, nome2: String, funcao:(i: String, j: String) -> String) : String {
  return funcao(nome1, nome2)
}
```

Exercício 4

```
fun soma(i: Int, j: Int) = i + j
fun cat(i: String, j: String): String = "${i}${j}"

fun main() {
  println(xpto(2, 3, ::soma))
  println(xpto("Jo", "ão", ::cat))
}

fun <T> xpto (x: T, y: T, callback: (x: T, y: T) -> T) : T = callback(x, y)
```

Exercício 5

```
// Funções de alta ordem
fun processaInteiro(i: Int, f: (Int) -> Int) = f(i)
fun processaInteiros(i: Int, j: Int, f: (Int, Int) -> Int) = f(i, j)

// Funções
fun soma(i: Int, j: Int) = i + j

fun multiplica(i: Int, j: Int) = i * j

fun raizQuadrada(numero: Int) = (1..numero).firstOrNull { it*it == numero } ?: -1

fun main() {
  // 1. Ivocação da funções sem Lambda
  println(processaInteiros(10, 5, ::soma))
  println(processaInteiros(10, 5, ::multiplica))
  println(processaInteiro(25, ::raizQuadrada))

  // 2. Invocação das funções com Lambda
  println(processaInteiros(10, 20) { n, m -> n + m })
  println(processaInteiros(10, 20) { n, m -> n * m })
  println(processaInteiro(10 ) { n -> (1..n).firstOrNull { it*it == n } ?: -1})
}
```