Kotlin

A Linguagem Kotlin

Tipos de Informação

A linguagem Kotlin classifica as informações nos seguintes tipos:

- Tipos numéricos
  - **Byte, Short, Int e Long** para inteiros

```
val x = 1 // Int
val b: Byte = 11
val num = 1L // Long
val valor = 1_000_000_000 // Long utilizando o separador "_"
val quantidade: Int? // Inteiro que aceita valor Nulo (null)
```

  - Float e Double para números de ponto flutuante

```
val y = 3.14 // Double
val taxa = 1.42f // Float
val z: Float // Float não inicializado
```

Tipos de Informação

- Textos
  - **String** para textos

```
val s1 = "Apóstrofes trabalham bem com literais string."
val s2 = "É \'fácil\' isolar o delimitador numa string."
val s3 = """
        Você pode criar
        multi-linhas de strings como esta.
"""
val s4 = """
        | Você pode criar
        | multi-linhas de strings como esta.
        """.trimMargin() // remove os espaços a esquerda
```

  - Char para caracteres

```
val c = '9' // Char
```

Tipos de Informação

- Lógicos
  - **Boolean** para tipos lógicos representados por *true* e *false*

```
val ganhou: Boolean = true
val valido = false
var ehMaior = 3 > 5;
var ehDivisivelPor2 = 10 % 2 == 0;
```

Tipos de Informação

SENAI-SP

- Coleções de dados
  - **Array** para vetores

```
val lista = arrayOf(1, 2, 3) // [1, 2, 3]
lista[1] = 1
val lista2 = IntArray(3) // [0, 0, 0]
val lista3 = IntArray(3) { it * 1 + 1 } // [1, 2, 3]
val lista4 = arrayOf(
 "Casa", "Mobilia","Plantas") // [ "Casa" , "Mobilia", "Plantas" ]
```

  - **List, MutableList e ArrayList** para coleções de dados

```
val lista5 = arrayListOf<Int>() // ArrayList de inteiros vazia
lista5.add(2) // adiciona 2 ao ArrayList
val lista6 = mutableListOf<Int>() // ArrayList de inteiros vazia
```

Tipos de Informação

- Coleções de dados
  - **Set** para conjuntos de ordenação indefinida

```
var halogenos = mutableSetOf(
  "flúor",
  "cloro",
  "bromo",
  "iodo",
  "astato"
) // Set de Strings

var valores = mutableSetOf<Double>( ) // Set de Double vazia
var num = HashSet<Int>( ) // Set de Int vazia
```

Tipos de Informação

- Coleções de dados
  - **Map** para conjuntos com chave associativa

```
var gifts = mutableMapOf(
  // Chave      Valor
  "primeiro" to "celular",
  "primeiro" to"carro",
  "primeiro" to"anel de ouro"
) // Map de String como chave e String como valor

var gasesNobres = buildMap<Int, String> {
  put(2, "helio")
  put(10, "neon")
  put(18, "argonio")
};
```

Tipos de Informação

- Data, Hora e intervalos de tempo
  - **DateTime** para Data e Hora

```
val agora = LocalDate.now()
val muroBerlin = LocalDate.of(1989, 11, 9)
val fmt = DateTimeFormatter.ofPattern("yyyy-dd-MM HH:mm:ss")
val pousoNaLua = LocalDate.parse("1969-07-20 20:18:04", fmt)
```

  - **Duration** para intervalos de tempo

```
var superMaratona = Duration.ofHours(1)
        .plusMinutes(3)
        .plusSeconds(2)
val feriado = Duration(88, ChronoUnit.HOURS)
val diferença = ChronoUnit.DAYS.between(
        muroBerlin, pousoNaLua)
```

Tipos de Informação

- Enumerações
  - **enum** para definir enumeração de informações

```
enum class Situacao {
  nada, executando, parado, pausado
}

enum class EstadoCivil {
  solteiro, casado, desquitado, divorciado, amasiado, viuvo
}

enum class NumerosPrimosAte60 {
  3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 41, 43, 47, 53, 59
}
```

Tipos de Informação

- Sinônimos de assinatura de funções
  - **typealias** para criar um “apelido" para outro tipo

```
typealias operacao = (Double, Double) -> Double

class Soma {
  companion object {
    const final simbolo = "+"
    val calcule: operacao = { n1, n2 -> n1 + n2 }
  }
}

class Multiplica {
  companion object {
    const final simbolo = "x"
    val calcule: operacao = { n1, n2 -> n1 * n2 }
  }
}
```

Tipos de Informação

- Tipo genérico
  - **Any** representa “qualquer" tipo

```
val mensagem = mutableMapOf<Int, Any> (
  1 to "sucesso na gravação",
  2 to 15.52,
  3 to LocalDate.now()
)
```

Tipos de Informação

SENAI-SP

- Tipo Função
  - **Function** descreve a “assinatura” de uma função

```
fun calc(
  no1: Double,
  no2: Double,
  op: (Double, Double) -> Double
) : Double {
  return op(no1, no2)
}

val multiplica = { vl1: Double, vl2:Double -> vl1 * vl2 }

main() {
  println(calc(2.0, 3.0) { v1, v2 -> v1 + v2 })  // resultado:  5.0
  println(calc(2.0, 3.0) {v1, v2 -> v1 - v2 })   // resultado: -1.0
  println(calc(2.0, 3.0, multiplica))            // resultado:  6.0
}
```

Tipos de Informação

- Tipo Função
  - Função com parâmetro nomeado, opcional e valor padrão

```
fun habilitaIndicadores(
  negrito: Boolean,
  oculto: Boolean = false
) : Unit  { . . . }

main() {
  habilitaIndicadores(oculto = true, negrito = true)
  . . .
  habilitaIndicadores(oculto = true)
  . . .
  habilitaIndicadores(true, true)
  . . .
  habilitaIndicadores(true)
}
```

Tipos de Informação

- Tipo Função
  - Função que retorna função

```
fun adiciona(valor: Int) : (Int) -> Int {
  return { total -> total + valor }
}

void main() {
  val valor1 = adiciona(4)
  val valor2 = adiciona(2)

  println(valor1(3))  // resultado:  7
  println(valor2(3))  // resultado:  5
}
```