**INTRODUÇÃO AO KOTLIN - RESUMO**

**Tipos de dados e conversores**

| **Tipos de dados** | **Conversores** |
| --- | --- |
| * Int * Long * Float * Double * Array * Boolean * Char * Byte * Short * Null | * toByte() * toShort() * toInt() * toLong() * toFloat() * toDouble() * toChar() |

**Variáveis**

* **Var(mutável, camelCase):** valor pode ser alterado durante o código.

| exemplo1:  1 var currentAge = 22 | exemplo2:  1 var currentAge:Int?  2 currentAge = null ou 22 |
| --- | --- |

* **Val(imutável, camelCase):** variável apenas com valor atribuído.

| exemplo1:  1 var currentAge = 22 | exemplo2:  1 var currentAge:Int?  2 currentAge = null ou 22 |
| --- | --- |

* **Const Val(imutável, SNAKE\_CASE:** constante cujo valor é atribuído durante a compilação.

exemplo:

1 const val MIN\_AGE = 16

2 const val MAX\_AGE = 68)

**Nulidade**

* Qualquer tipo pode ser nulo, mas deve ser explicitado na declaração da variável através do uso do sinal de interrogação(?);
* A inferência de tipo não atribui nulidade.

**Operadores aritméticos básicos**

* Os operadores podem ser chamados tanto como expressão quanto como comando. O resultado será o mesmo;
* A função de soma também funciona para concatenar Strings.

| **FUNÇÃO** | **EXPRESSÃO** | **COMANDO** | **ATRIBUIÇÃO** |
| --- | --- | --- | --- |
| soma | a + b | a.plus(b) | a += b |
| subtração | a - b | a.minus(b) | a -= b |
| multiplicação | a \* b | a.times(b) | a\*= b |
| divisão | a / b | a.div(b) | a/=b |
| resto | a % b | a.mod(b) | a%=b |

**Operadores Comparativos básicos**

* Os comandos **compareTo** retornam os **valores -1(menor que), 0(igual) ou 1(maior).** Já os **operadores** retornam **valores booleanos;**
* O comando **equals** retorna um **booleano**.

| **FUNÇÃO** | **EXPRESSÃO** | **COMANDO** |
| --- | --- | --- |
| maior/menor | a > b ou a < b | a.compareTo(b) > 0  ou  a.compareTo(b) < 0 |
| maior/menor  igual | a >= b ou a<= b | a.compareTo(b) >= 0  ou  a.compareTo(b) =< 0 |
| Igual | a == b | a.equals(b) |
| diferente | a != b | !(a.equals(b)) |

**Operadores Lógicos**

* Quando se utiliza o comando, é recomendado colocar a expressão entre parênteses.

| **FUNÇÃO E EXPRESSÃO** | **COMANDO** |
| --- | --- |
| E (&&) | (expressão 1) and (expressão2) |
| Ou (| |) | (expressão 1) or (expressão2) |

**Operadores In e Range**

| **FUNÇÃO E EXPRESSÃO** |
| --- |
| contém (In) |
| Não contém (!In) |
| range/ Faixa de valores (Int..Int) |

* Se valor está presente em uma lista ou uma faixa(range) de valores;
* Range cria um intervalo de valores que inicia no primeiro parâmetro e acaba no segundo.

exemplo de **in:**

1 val numbers = lisOf(3,9,0,1,2)

2 println(12 **in** numbers) //false

exemplo de **range:**

1 println(12 in 0..20) //true

**Manipulação de Strings**

* Strings possuem diversos métodos associados(indexação, concatenação, formatação);

Indexação

* String também é tratada como um array de Char;
* First(), last(), String.length, String[index];

Concatenação

* Para concatenar duas strings o plus/+ pode ser utilizado;
* Para concatenar uma variável a uma String, os símbolos${} devem ser inseridos;

Formatação

| NOME | FUNÇÃO | MÉTODOS |
| --- | --- | --- |
| Capitalização de  strings | Alterar a formatação entre letras maiúsculas e minúsculas | capitalie(), toUpperCase(), toLowerCase(), decapitlized() |
| Remoção de espaços | Remove espaços vazios e características inadequadas para impressão | trimEnd(), trimStart(), trim() |
| Substituição de caracteres | Substituir caracteres por outros | replace(x,y) |
| formatação | Formatar outros valores para um padrão de string | “padrão${valor}”.format(valor) |

**Empty x Blank**

* Métodos de comparação;
* String está vazia, em branco ou é nula?
* Se o tamanho da string for 0 está **empty e blank**.

exemplo:

1 val string = “”

* Se o tamanho for > 0 mas todos os caracteres são espaços em branco, está **blank** mas não empty.

exemplo:

1 val string = “ “

**Introdução a funções**

* Prefixo **fun nomeDaFuncao(nome:Tipo):TipoRetorno{}**
* Funções anônimas, single-line, inline, extensões, Lambdas, ordem superior;

Funções de ordem superior

* Recebem outra função ou lambda por parâmetro;
* Bastante úteis para a generalização de funções e tratamento de erros;

exemplo:

1 val x = calculate(12, 4m :: sum)

2 val y = calculate(12,4){ a,b -> a\*b }

Funções single-line

* Prefixo Fun nomeDaFuncao(nome:Tipo)=retorno;
* Função de uma única linha;
* Infere o tipo de retorno;

exemplo:

1 private fun getFullName(name:String, lastName:String) = “$name $lastname”

2

Funções/ extensões

* Prefixo **fun Tipo.nomeDaFuncao()**;
* Cria uma função que só pode ser chamada por um tipo específico, cujo valor pode ser referenciado dentro da função através da palavra **this**;

exemplo:

1 fun String.randomCapitalizedLetter ( ) =

2 this[(0..this.length-1).random( )].toUpperCase( )

**Estruturas de controle**

* if/else, when, elvis operator;
* Pode ser utilizado tanto para controle quanto para atribuição;
* Pode ser encadeado com múltiplas estruturas.

exemplo: when

1 when {

2 case1 -> { }

3 case2 -> { }

4 case3 -> { }

5 else -> { }

6 }

exemplo: elvis operator

1 val a:Int? = null

2 var number = a ?: 0

**Atribuição**

* O valor atribuído tem que estar na última linha do bloco;
* A condicional pode não usar chaves se só fizer a atribuição.

When

* Equivalente ao switch de outras linguagens;
* Aceita tanto valores quanto condicionais;
* Aceita range como case;

Elvis Operator

* O mais próximo que a linguagem possui de um operador ternário;
* Verifica se um valor é nulo e apresenta uma opção caso seja;
* Pode ser encadeado.

**Estruturas de repetição**

* While, do..while, for e forEach;
* Estruturas similares às convencionais em outras linguagens;
* Aceita os comandos in, range, until, downTo e step.

For

* In: conta do valor inicial até o valor final apresentado;
* Until: conta do calor atual até o calor apresentado menos 1;
* DownTo: conta de maneira decrescente;
* Step:determina o intervalo da contagem;