**Resumo Kotlin**

**Tipos de Dados:**

* Int
* Long
* Float
* Double
* Array
* Boolean
* Char
* Byte
* Short
* Null!

**Exemplos:**

\\obter valor máximo que os tipos de dados podem receber

println(Int.MAX\_VALUE)

println(Float.MAX\_VALUE)

println(Long.MAX\_VALUE)

println(Byte.MAX\_VALUE)

println(Short.MAX\_VALUE)

Ex:

/\*\*

\* You can edit, run, and share this code.

\* play.kotlinlang.org

\*/

fun main() {

println(Int.MAX\_VALUE)

println(Float.MAX\_VALUE)

println(Long.MAX\_VALUE)

println(Byte.MAX\_VALUE)

println(Short.MAX\_VALUE)

println(Double.MAX\_VALUE)

}

**Conversão de Dados:**

Para converter um dado para chamar a função .to.

.toByte()

.toShort()

.toInt()

.toLong()

.toFloat()

.toDouble()

.toChar()

/\*\*

\* exemplo de conversão de dados int para string

\*/

fun main() {

var currentYear = "que ano hein!"

currentYear = 2021.toString() + " vai ser melhor!"

println(currentYear)

}

**Declaração de variável**

* **var:** valor mutável, CamelCase;

//o nome de uma variável sempre será iniciado com uma letra minúscula e sem espaço, porém a partir da segunda palavra será CamelCase, ou seja, iniciará com letra maiúscula.

* **val:** valos imutável, CamelCase - terá sempre o valor atribuído (similar ao final do Java);
* **const val:** valor imutável, SNAKE\_CASE (receberá o valor durante a compilação);

//SNAKE\_CASE ou seja, seu nome será todo em maiúsculo.

Obs.: se uma constante estiver dentro da função main, obrigatoriamente ela será um **val**, porém se estiver fora será um **const val**.

**var** currentAge = 22 - pode receber apenas números inteiros

**var** currentAge:Int? - pode receber tanto números inteiros quanto valor nulo

\\camelCase e valor definido e alterado durante a execução;

**val** currentAge = 22 - pode receber apenas números inteiros

**val** currentAge:Int? - pode receber tanto números inteiros quanto valor nulo

\\camelCase e valor definido e não alterado durante a execução;

**const val** MIN\_AGE = 16

**const val** MAX\_AGE = 68

\\camelCase e valor definido durante a compilação (desenvolvimento);

Obs.:

Uma variável não poder ser declarada sem tipo de dado e sem atribuição;

Uma variável com inferência de tipo só receberá valores do mesmo tipo;

**Valor nulo e Operadores aritméticos básicos:**

/\*\*qualquer tipo pode ser nulo, porém isso deve ser explicitado na declaração de variável através do uso da interrogação (?);\*/

/\*\*A inferência de tipo não atribui nullability;\*/

Ex:

var mounth:Int? = null

//atribui corretamente null para a variável mês

var day:Int = null

//Erro de compilação devido a não utilização do ?

**Comandos Aritméticos**

| **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 |

Obs.: operador **+** também serve para concatenar strings

**Operadores Comparativos**

| **Função** | **Expressão** | **Comando** |
| --- | --- | --- |
| maior/menor | a > b ou a < b | a.compareTo(b) > 0  ou  a.compareTo(b) < 0 |
| maior/menor igual | a >= ou a <= b | a.compareTo(b) >= 0  ou  a.compareTo(b) <= 0 |
| igual | a ==b | a.equals(b) |
| diferente | a != b | !(a.equals(b)) |

Obs.:

Os comandos compareTo retornam os valores -1 (menor que), 0 (igual) ou 1 (maior que).

Os operadores >, <, == retornam valores booleanos.

O comando equals retornam valores booleanos.

**Ex:**

/\*\*

\* função principal

\*/

fun main() {

val d=22

val t=90

println(d>t) //retorna false

println(d.compareTo(t)) //retorna -1 que equivale a d<t

}

**Eliminando números mágicos (sem significado)**

//eliminando números mágicos (sem significado)

const val EQUAL =0

const val LESS =-1

const val MORE =1

//função principal

fun main() {

val d=22

val t=90

println(d>t) //retorna false

println(d.compareTo(t) == MORE) //retorna -1 que equivale a d<t

}

**Operadores Lógicos**

| **Função e Expressão** | **Comando** |
| --- | --- |
| e (&&) | (expressão1) and (expressão2) |
| ou (||) | (expressão1) or (expressão2) |

Obs.: Quando se utiliza os comandos recomenda-se colocar as expressões entre parênteses.

**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;

fun main() {

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

println(12 in numbers) //false

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

}

**Ex:**

fun main() {

val bingo = listOf(8,6,34,2,13)

//var number = (1..34).random()//.random gera um número aleatório entre o intervalo //especificado

var number = 34

println(number)

println(number in bingo)

}

--

const val MIN\_AGE = 16

const val MAX\_AGE = 68

////verificar se a idade informada está dentro do intervalo/range

fun main() {

var age = (10..100).random()

println(age)

println(age in MIN\_AGE..MAX\_AGE) //verifica se age está entre o intervalo e se atende a duas condições

println(age >= MIN\_AGE && age <= MAX\_AGE) //verifica se age está entre o intervalo e se atende a duas condições

}

**Manipulação de Strings**

**Indexação**

* String como array;
* First(), last(), String.length, String[index];

//função principal

fun main() {

val s = "Olá, mundo!"

println(s[0]) //acessa a primeira posição

println(s.first()) //acessa a primeira posição

//imprime 0

println(s[s.length-1]) //acessa a última posição

println(s.last()) //acessa a última posição

//Imprime !

}

**Concatenação**

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

**Ex:**

//função principal

fun main() {

val name = "Ana"

val s = "Olá"

println(s + name)

//imprime OláAna

println("${s}, ${name}!")

//imprime Olá Ana

println("Bem vinda(o), $name!")

//imprime Bem vinda(o), Ana!

}

--

//função principal

fun main() {

val welcome = "Bem vindo(a)"

var name = "luisa"

println("$welcome, ${name.capitalize()}!")//.capitalize() torna a primeira letra da string maiúscula;

//${} deve ser usado quando há tratamento do resultado (ex. espaço)

}

**Formatação (Métodos)**

| Nome | função | Métodos |
| --- | --- | --- |
| Capitalização de strings | Alterar a formatação entre letras maiúsculas e minúsculas | capitalize(), toUpperCase(),  ToLowerCase(), and decapitalize() |
| Remoção de espaços | Remove espaços vazios e caracteres inadequados para impressão | trimEnd(), trimStart(), trim() |
| Substituição de caracteres/trechos/valor de uma variável pelo valor de outra | Substituir caracteres por outros | replace(x,y) |
| formatação | Formatar outros valores para um padrão de string | “padrão ${valor}”.format(valor) |

**Diferença entre Empty e Blank**

* métodos de comparação;
* string está vazia, em branco ou nula;

fun main() {

val s=""

println(s.isEmpty())

//true

println(s.isBlank())

//true

println(s.isNullOrBlank() ||

s.isNullOrEmpty())

//true

}

--

fun main() {

val s=" "

println(s.isEmpty())

//false

println(s.isBlank())

//true

}

* Se tamanho da string (s.length) for 0 está empty e blank;
* Se tamanho for > 0, mas todos os caracteres são espaços em branco ela está blank e não empty;

fun main() {

val empty=""

println(empty.length)//retorna o tamanho de empty

val blank=" "

println(blank.length)//retorna o tamanho de blank

println(empty.isEmpty() && empty.isBlank())

//retorna true se empty atender as duas condições

println(blank.isEmpty() || blank.isBlank())

//retorna true se empty atender a pelo menos uma das condições

}

**Introdução a funções:**

* Prefixo **Fun nomeDaFunção(nomepar:Tipopar):TipoRetorno{}**
* Funções anônimas, single-line, inline, extensões, Lambdas, ordem superior;
* **Fun nomeDaFunção(nomepar:Tipopar)**//sem retorno (void)
* **Fun nomeDaFunção():TipoRetorno{}**//sem parâmetro, colocar ()

**Simplificando as funções:**

**Ex 1:**

private fun getFullName(name:String, lastName:String):String{

val fullname= “$name $lastName”

return fullname

}

**Ex 2:**

private fun getFullName(name:String, lastName:String):String{

return “$name $lastName”

}

**Ex 3:**

private fun getFullName(name:String, lastName:String) = $name $lastName

**Funções de ordem superior:**

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

val x = calculate(12,4,::sum)//função por parâmetro

val y = calculate(12,4){a,b -> a\*b }//lambda por parâmetro

**Ex 1:**

//função principal

fun main() {

val z:Int

z = calculate(34,90,::sum)

println(z)

}

fun sum(a1:Int,a2:Int) = a1.plus(a2)

fun calculate(n1:Int,n2:Int,operation:(Int,Int)->Int):Int{

val result = operation(n1,n2)

return result

}

**Ex 2:**

//função principal

fun main() {

val z:Int

z = calculate(34,90){a,b -> a\*b}//multiplicação

println(z)

}

fun sum(a1:Int,a2:Int) = a1.plus(a2)

fun calculate(n1:Int,n2:Int,operation:(Int,Int)->Int):Int{

val result = operation(n1,n2)

return result

}

**Funções single-line**

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

private fun getFullName(name:String, lastName:String) = $name $lastName

private fun getFullName(name:String, lastName:String):String{

return “$name $lastName”

}

**Funções/extensões**

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

fun String.ramdomCapitalizedLetter() =

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

**Estrutura de Controle**

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

if(expressão){

//bloco de código

}else if (expressão2){

//bloco de código

}else{

//bloco de código

}

--

when {

case1 -> {}

case2 -> {}

case3 -> {}

else -> {}

}

--

val a:Int? = null

var number = a ?: 0 //validação para ver se a é nulo. Se sim mostra 0

**Atribuições, When e Elvis operator**

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

val maxValue = if (a > b) a else if (a < b) b else b

val minValue = if (a > b){

println(“b( $b) é o menor valor”)

b

}else if (a > b){

println(“a( $a) é o menor valor”)

a

}else{

println(“os valores são iguais”)

b

}

**When**

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

when {

a > b -> {}

a != b && a > c -> {}

b == 0 -> {}

else -> {}

}

--

when(year) {

-4000.. 475 -> //Antiguidade

476.. 1452 -> //Medieval

1453.. 1789 -> //Moderna

currentYear -> //ano atual

}

**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;

**Ex 1:**

val a:Int? = null

val c:Int? = 9

var number = a?: b?: 0 //encadeamento

//quando apenas b for nulo recebe a

//quando apenas a for nulo recebe b

//quando a e b forem nulos recebe 0

**Ex When:**

//função principal

fun main() {

val grade = (0..10).random()

println(grade.getStudentStatus())

}

fun Int.getStudentStatus():String{

println("nota $this") //this referência o Int

return when(this){ //retornando direto uma atribuição sem a necessidade de declarar variável

in 0..4 -> "Reprovado"

in 5..7 -> "Mediano"

in 8..9 -> "Bom"

10 -> "Excelente"

else -> "Indefinido"

}

}

**Ex Elvis Operator:**

**Ex 1:**

//função principal

fun main() {

var t:Int

var x:Int? = null

var w:Int? = 10

t = x?:w?: -1

println(t)

}//retornou 10

**Ex 2:**

//função principal

fun main() {

var t:Int

var x:Int? = null

var w:Int? = null

t = x?:w?: -1

println(t)

} //retornou -1

**Ex 3:**

//função principal

fun main() {

var t:Int

var x:Int? = 40

var w:Int? = null

t = x?:w?: -1

println(t)

} //retornou 40

**Estrutura de Repetição**

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

while(Condição){

}

--

do{

//bloco

}while(Condição)

--

for(i in 0..20 step 2){ //conta de 0 a 20 de 2 em 2

println(i)

}

**FOR**

* for(variavelIndexadora in/until/downTo faixa de valores/condicional step intervalo)
  + in: conta do valor inicial até o valor final estabelecido;
  + until: conta do valor atual até o valor estabelecido menos 1;
  + downTo: conta de maneira decrescente;
  + step: determina o intervalo da contagem;

**Exemplos For:**

for(1 in 0..20 step 2){

println(1)

}

--

for(i in 10 downTo 0){

println(i)

}

--

for(i in 0 until 10){

println(i)

}

--

var text = “Kotlin”

for (letter in text){

println(letter)

}

//letter imprime a string sendo 1 caracter por linha