Desenvolvimento para dispositivos móveis

Introdução à linguagem Kotlin

Professor Msc. Fabio Pereira da Silva
E-mail: fabio.pereira@faculdadeimpacta.com.br

Introdução ao Kotlin

- Desde Google I/O 2017 o desenvolvimento de aplicativos para Android suportam a linguagem Kotlin.
- Deixa o desenvolvimento mais produtivo
- Desenvolvida pela JetBrains, mesma do Android Studio
- Sintaxe moderna, expressiva, simples e agradável
- Compilada para executar na JVM
  - Interoperabilidade total com Java

Introdução ao Kotlin

- Exemplo, comparando Java com Kotlin para clicar em um botão no app:
- Em Java

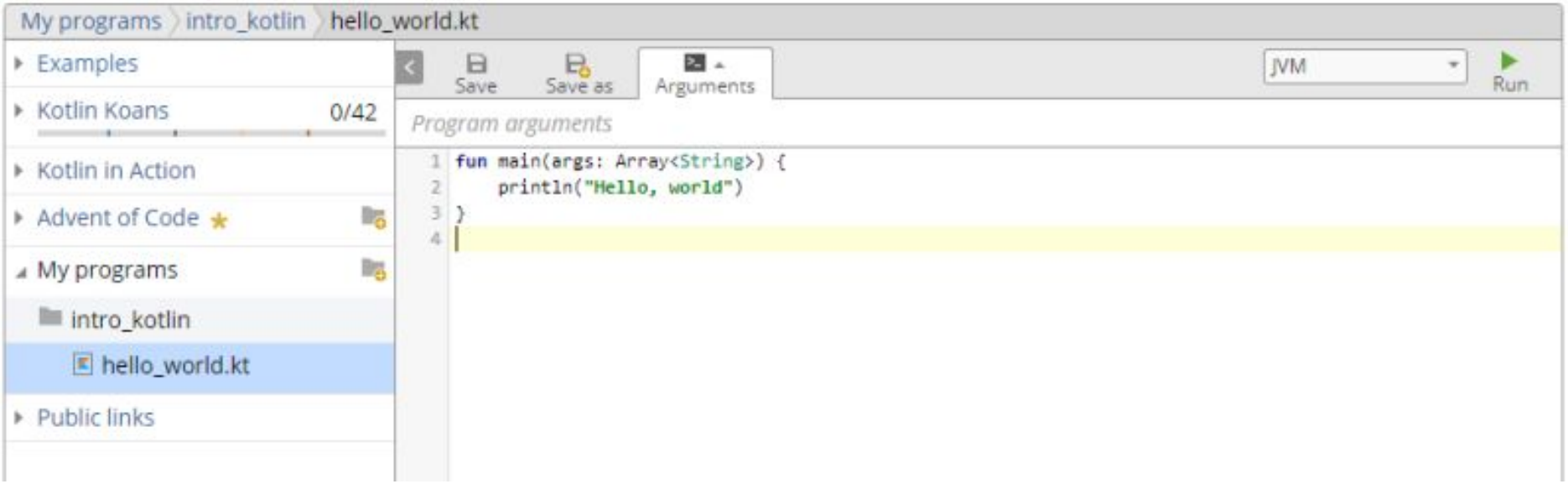
```
View btClicar = findViewById(R.id.btClicar);
btClicar.setOnClickListener(new View.OnClickListener() {
    @Override public void onClick(View v) {
        clicarBotao()
} });
```

- Em Kotlin

```
findViewById<Button>(R.id.btClicar).setOnClickListener(clicarBotao());
```

Treinar Kotlin

- Acesse https://try.kotlinlang.org/ para testar alguns exemplos e se familiarizar com a linguagem .
- Crie uma conta para poder criar e salvar seus projetos.
- A documentação do Kotlin está disponível em:



Variáveis

- O armazenamento de informações pelo computador em sua memória, se dá em uma região nomeada através de uma variável
- Uma variável possui:
  - NOME
  - TIPO
  - CONTEÚDO
- As regras para nomes de variáveis mudam de uma linguagem para outra

Variáveis

- Variáveis devem ser declaradas antes de serem utilizadas
- Ao declarar uma variável, o computador reserva um espaço na memória para ela
- A memória é constituída de bytes, que são conjuntos de 8 bits
- Cada tipo de variável ocupa um tamanho diferente na memória, isso varia para cada linguagem de programação

Variáveis

- Elas contêm dados temporários
- Podemos pensar nas variáveis como “células do Excel” armazenadas na memória RAM.
- Embora não seja obrigatório, é uma boa prática de programação definir o tipo de informação que as variáveis armazenarão.
- Isto é chamado “declarar uma variável”
- O tipo de informação que podemos armazenar numa variável depende do tipo de dados escolhido para aquela variável

Primeira aplicação em Kotlin

- Um programa Kotlin tem a extensão .kt
- HelloWorld – hello_world.k

```
fun main(args: Array<String>) {
 println("Hello, world")
}
```

Sintaxe básica:

– fun: define uma função

– Tipo do parâmetro especificado depois do nome , separado por : (args: Array)

– Não precisa de: Bloco definido por chaves {}

Funções print e println

- Imprimir no console
  - print(String): sem quebra de linha
  - println(String): com quebra de linha

```
fun main() {
print("Hello, world")
println("Hello, world")
println("Kotlin")
}
```

Tipos Básicos de Dados

- Dados Numéricos Inteiros
  - São os números positivos e negativos sem casas decimais
- Dados Numéricos Reais
  - São os números positivos e negativos que possuem casas decimais
- Dados Literais (caracteres)
  - São sequências de caracteres
- Dados Lógicos ou Booleanos
  - Podem ser verdadeiros ou Falsos, apenas

Tipos Básicos de Dados

- // Tipos de dados no Kotlin
- fun main(args: Array<String>) {
- var numero: Double = 3.554525635425
- var numero2: Float = 5.3f
- var numeroInteiro: Int = 10
- var numeroLong: Long = 100000000L
- var resultado: Boolean = false
- **Exercício: Imprima o valor de cada uma das variáveis acima, uma em cada linha**

Operadores aritméticos

Operador	Descrição
+	Somar
-	Subtrair
*	Multiplicar
Divisão	/
Módulo	(%) – Resto da divisão

Exercícios

- 1) Inicialize duas variáveis, realize a soma e apresente o resultado
- 2) Inicialize duas variáveis, realize a subtração e apresente o resultado
- 3) Inicialize duas variáveis, realize a multiplicação e apresente o resultado
- 4) Inicialize duas variáveis, realize a divisão e apresente o resultado
- 5) Inicialize duas variáveis, calcule e apresente o resultado da divisão da maior pela menor

Soma de valores

```
// Operadores aritmeticos
fun main() {
 /*
Somar (+)  Subtrair (-) Multiplicar (*) Divisao (/) Módulo (%) - Resto de um divisão*/
var numero1 = 202
var numero2 = 5
var resultado = numero1 + numero2
println( resultado )
}
```

Operadores relacionais

Operador	Descrição
==	Igual a
!=	Diferente
>	Maior que
<	Menor que
>=	Maior ou igual
<=	Menor ou igual

Operadores lógicos

- Os operadores lógicos são usados para representar situações lógicas que não podem ser representadas por operadores aritméticos.
- Também são chamados conectivos lógicos por unirem duas expressões simples numa composta. Podem ser operadores binários, que operam em duas sentenças ou expressões, ou em uma única sentença.

Operadores lógicos

- Operador AND

<expr_aritmética_01>	<expr_aritmética_02>	<expr_aritmética_01> && <expr_aritmética_02>
verdadeiro	verdadeiro	verdadeiro
verdadeiro	falso	falso
falso	verdadeiro	falso
falso	falso	falso

<expr_aritmética_01>	<expr_aritmética_02>	<expr_aritmética_01> || <expr_aritmética_02>
verdadeiro	verdadeiro	verdadeiro
verdadeiro	falso	verdadeiro
falso	verdadeiro	verdadeiro
falso	falso	falso

Operadores lógicos

Operador	Descrição
&&	E
||	Ou

Exercícios

- 6) Inicialize duas variáveis com as notas da notaProfessor e da notaProvaGeral do aluno. Apresente em console a seguinte condição. Se uma delas for maior do que 6, a operação deve retornar true, caso contrário deve retornar false.
- 7) Inicialize duas variáveis com as notas da notaProfessor e da notaProvaGeral do aluno. Apresente em console a seguinte condição. Se as duas forem maiores do que 6 a operação deve retornar true, caso contrário deve retornar false.

Resolução – Exercício 6

```
fun main() {
var notaProfessor = 5//0-10
var notaProvaGeral = 5 //0-10
println( notaProfessor > 6 || notaProvaGeral >= 6 )
}
```

Resolução – Exercício 7

```
fun main() {
var notaProfessor = 5//0-10
var notaProvaGeral = 5 //0-10
println( notaProfessor > 6 && notaProvaGeral >= 6 )
}
```

Estruturas de controle

- As estruturas de controle determinam o curso de ações de um algoritmo ou programa.
- A lógica do procedimento flui através das instruções da esquerda para a direita e de cima para baixo.
- As instruções de controle, ou seja, os comandos que controlam a tomada de decisões e as iterações podem alterar a ordem de execução das instruções.

Estrutura de decisão simples

- Testa uma condição única e executa uma instrução ou um bloco de instruções.
- Sintaxe: **If { condição }**

```
fun main() {
var idade = 18
if( idade>=18){
println("Adulto")
}
}
```

Estrutura de decisão composta

Testa mais de uma condição e executa um dos vários blocos de instruções.

Sintaxe:

```
if condição {
Instruções
} Else If condição {
Instruções
}
Else{
Instruções
 }
```

Estrutura de decisão composta

```
fun main(){
 var idade = 18
 if( idade < 14 ){
 println("Criança") }
else if( idade>=14 && idade < 18 ){
println("Adolescente")
}else{
println("Adulto")
}
}
```

Exercício

- 8) Inicialize 3 temperaturas e apresente em console se alguma delas é negativa
- 9) Inicialize as notas dos alunos, N1 e N2. Verifique se a média das duas é maior do que 6. Se sim, exiba uma mensagem indicando que o aluno foi aprovado. Senão Inicialize uma nova nota N3 e verifique se o aluno foi aprovado.

Resolução – Exercício 8

```
fun main(){
var temp1:Double = -1.0
var temp2:Double = 3.0
var temp3:Double = 4.0

if(temp1<0 || temp2<0 || temp3<0){
print("Uma das temperaturas é negativa")
}
else{
print("Todas as temperaturas são positivas")
}
}
```

Resolução – Exercício 9

```
fun main(){
var n1:Double = 3.0
var n2:Double = 3.0
var media:Double = (n1+n2)/2
if(media > 6){
println("Aprovado") }
else{ var n3:Double = 10.0
media = (media + n3)/2
if(media>6){
println("Aprovado")
}
else{
println("Reprovado")
println(media) }}}
```

Estruturas de controle – seleção de múltipla escolha

- when (opcao) — testa uma condição única e executa um dos vários blocos de instruções.
- Exemplo:

```
fun main(){
var opcao = 1
var resultado = when (opcao){
1 -> "Café puro"
2-> "Café com Leite"
else -> "Erro"
}
println(resultado)
}
```

Estruturas de controle – seleção de múltipla escolha

```
fun main(){
    var opcao = 3  // para controlar o menu
    var dado = if(opcao==1) 5 else -1
    var resultado =   when (opcao){
        1 -> println("Café puro!")
        2 -> println("Café com Leite")
        3 -> {
                println("Leite")
                println("Da promoção")
                 }
        else -> {
                        println("Erro")
                        println("escolha uma opção válida")
                        }}
    println(resultado)}
```

Estruturas de Controle

- ESTRUTURA SEQUENCIAL
- ESTRUTURAS CONDICIONAIS
  - Estrutura Condicional Simples
  - Estrutura Condicional Composta
  - Estrutura de Decisão Encadeada
  - Seleção entre duas ou mais Sequências de Comandos
- ESTRUTURA DE REPETIÇÃO
  - Repetição com Teste no Início
  - Repetição com Teste no Final

Estrutura de repetição

Digamos que o usuário deseja escrever automaticamente uma sequência numérica de 1 a 10, com um número em cada linha. O algoritmo ficaria extenso mesmo para algo tão simples.

```
1  algoritmo "numeros"
2  var
3  inicio
4  escreval('1')
5  escreval('2')
6  escreval('3')
7  escreval('4')
8  escreval('5')
9  escreval('6')
10 escreval('7')
11 escreval('8')
12 escreval('9')
13 escreval('10')
14 fimalgoritmo
```

Estruturas de repetição

- Observe também que o comando **escreval** se repete diversas vezes, mudando apenas o valor dentro do parênteses.
- As estruturas de repetição ajudam ao programador a tratar códigos repetitivos com poucas linhas.
- A seguir vamos aprender como usá-las.

Estruturas de repetição

- **Uma estrutura de repetição obrigatoriamente possui:**
  - ***Uma variável de controle.***
    - Usada para contar quantas vezes o laço se repete.
  - ***Um incremento.***
    - Usado para aumentar ou diminuir o valor da variável de controle. Pode ser por atribuição ou por digitação do usuário.
  - ***Um teste lógico.***
    - Usado para verificar se a condição de parada foi atingida.

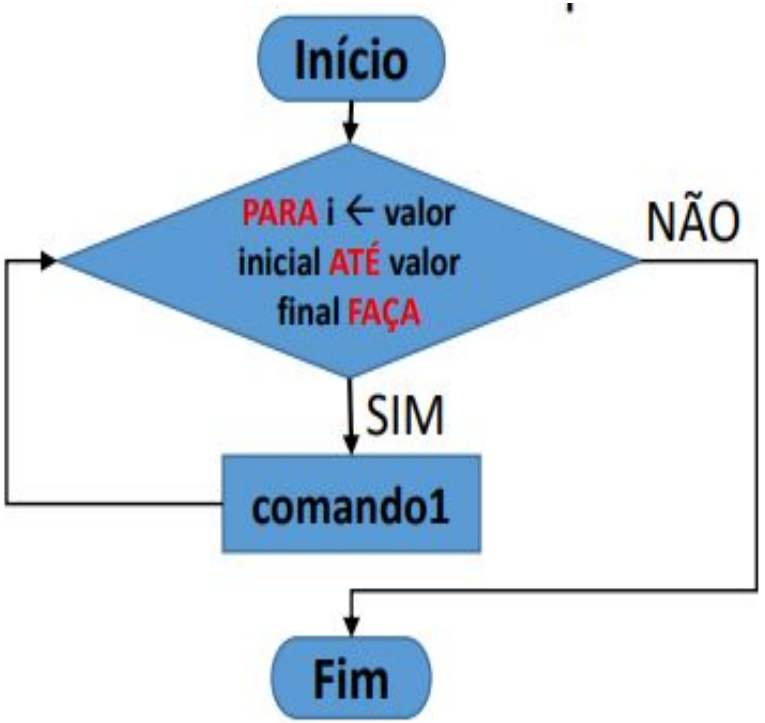
Estruturas de repetição

- Permite que uma sequencia de comandos seja executada repetidamente até que uma determinada condição seja satisfeita.
- Trechos de algoritmos e consequentemente comandos de um determinado programa que precisam ser repetidos para realizar algum tipo de leitura de dados ou cálculo aritmético são chamados de laços de repetição.

Estruturas de repetição - Para

- Estrutura de repetição para número definido de repetições (estrutura para): Essa estrutura de repetição é utilizada quando se sabe o número de vezes em que um trecho do algoritmo deve ser repetido.

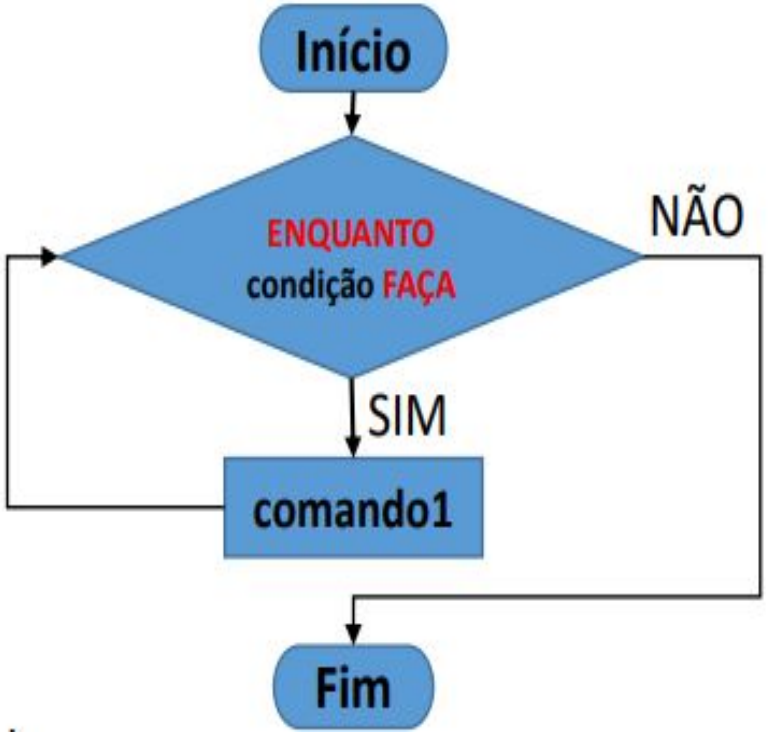
```
ALGORITMO
DECLARE
PARA i ← valor inicial ATÉ valor final FAÇA
comando1
FIM_ALGORITMO
```



Estruturas de repetição - Enquanto

- Estrutura de repetição para número indefinido de repetições e teste no início (estrutura enquanto): Essa estrutura de repetição é utilizada prioritariamente quando não se sabe o número de vezes em que um trecho do algoritmo deve ser repetido.

```
ALGORITMO
DECLARE
ENQUANTO condição FAÇA
comando1
FIM_ALGORITMO
```



Estruturas de repetição - While

```
// Comandos de repetição
fun main(){
var i=0
while (i<10){
if (i<9){
print("$i - ")
}
else {
println(i)
}
i++
}}
```

Estruturas de repetição - For

```
// Comandos de repetição
fun main(){
 for (valor in 1..10){       println(valor)
 }
}
```

Exercícios

- 10) Apresente todos os números pares entre 1 a 100, utilizando o FOR
- 11) Apresente todos os números ímpares entre 1 a 100 utilizando o While
- 12) Percorra todos os números de 1 a 100 e apresente quais números são divisíveis por 5.
- 13) Inicialize uma variável e realize o cálculo do seu fatorial

Resolução – Exercício 10

```
// Comandos de repetição
fun main(){
for (valor in 1..1000){
if(valor % 2 == 0){
println(valor)
}
}
}
```

Resolução – Exercício 11

```
// Comandos de repetição
fun main(){
    var i=1
    while (i<100){
        if(i%2==1){
         println(i)
        }
            i++
        }
    }
```

Resolução – Exercício 12

```
// Comandos de repetição
fun main(){
    var i=1
    while (i<=100){
      if(i%5==0){
        println(i)
      }
           i++
       }
    }
```

Resolução – Exercício 13

```
// Comandos de repetição
fun main(){
    var valor:Int = 5
    var decremento:Int = 0
    var fatorial:Int = 1

    decremento = valor
    while (decremento>=1){
      fatorial = fatorial * decremento
            decremento = decremento-1
       }
    println("O fatorial de, $valor, é, $fatorial")
    }
```

Modularização de algoritmos

- Em geral, um programa é executado linearmente, uma linha após a outra, até o fim.
- Entretanto, quando são utilizados sub-algoritmos, é possível a realização de desvios na execução natural dos programas.
- Assim, um programa é executado linearmente até a chamada de um sub-algoritmo.
- O programa que chama um sub-algoritmo (“chamador”) é temporariamente suspenso e o controle é passado para o sub-algoritmo, que é executado .
- Ao terminar o sub-algoritmo, o controle retorna para o programa que realizou a chamada (“chamador”).

Modularização de algoritmos

- Característica fundamental da programação:
  - Modular a resolução do problema através da sua **divisão em subproblemas** menores e mais simples;
  - Neste processo, cada subproblema pode ser analisado de forma individual e independente dos demais.

- Objetivo:
  - Facilitar o trabalho com problemas complexos;
  - Permitir a reutilização de módulos.
    - Com a modularização de um programa, as partes que o compõem podem ser desenvolvidas por diferentes equipes.

Modularização de algoritmos

- Refinamentos Sucessivos (top-down):
  - Divisão do problema inicial em subproblemas, e estes em partes ainda menores, sucessivamente, até que cada parte seja descrita através de um algoritmo claro e bem-definido.
    - Um algoritmo que resolve um determinado subproblema é denominado subalgoritmo.

Modularização de algoritmos

- A subprogramação é uma ferramenta que contribui com a tarefa de programar:

  - Favorecendo a estruturação do programa;
  - Facilitando a correção do programa;
  - Facilitando a modificação do programa;
  - Melhorando a legibilidade do programa;
  - Divisão do problema a ser resolvido em partes (modularização).

Subalgoritmos (Módulos)

- Por convenção, um subalgoritmo deve ser declarado acima dos módulos que o chamam;
- Todo subalgoritmo tem por objetivo a resolução de um determinado subproblema;
- Portanto, mantém as mesmas características de um algoritmo comum:
  - Pode ter dados de entrada;
  - Dados de saída; e
  - Conter qualquer tipo de comando aceito por um algoritmo.

Subprogramação

- As linguagens de programação oferecem algum tipo de suporte à subprogramação.
- Exemplos:Algol: bloco;
  - FORTRAN: subrotina;
  - Modula: co-rotinas;
  - ADA: tarefas;
  - C: funções;
  - Visualg: procedimentos e funções.
  - Pascal: procedimentos e funções.

Funções

- Sintaxe básica

```
fun nomeFuncao(param1: Tipo, param2: Tipo, …): TipoRetorno {
        // Corpo da função
}
```

- Exemplo:

```
fun main(args: Array<String>) {
var nome = “Fabio"
imprimir(nome)
val soma = somar(2, 3)
imprimir("Soma: $soma")
}
// recebe uma string e não retorna nada (Unit)
fun imprimir(s: String): Unit {
println(s)
}
// Recebe 2 inteiros e retorna uma inteiro
fun somar(a: Int, b: Int): Int {
return a + b
}
```

- Unit identifica que a função não retorna nada. Seu uso é opcional

Exemplo de Funções

```
// Exemplo de uso de valores padrão para parâmetros

fun escreve(nome:String, sNome:String, faculdade:String){

println("<<< $nome")

println("<<< $sNome")

println("<<< $faculdade")

}

fun main(){

 escreve("José","Silva","GV")

}
```

Objetos nulos (Null Safety)

- Em Kotlin não é possível armazenar valores nulos em variáveis e objetos, por padrão
- A forma de fazer isso é explicitar que pode receber nulo.
- Para receber nulo, a variável deve ter tipo e o operador ?

```
fun main(){
var nome:String? = "Fabio"
println("Olá $nome")
nome = null // OK
println("Olá $nome") }
```

Objetos nulos (Null Safety)

- Se uma variável nula for chamada sem verificação, o código não vai nem compilar.

- É preciso verificar antes se a variável é nula

```
fun main() {
var nome:String? = "Fabio"
println("Olá $nome")
nome = null // OK
println("Olá $nome")
if (nome != null){
println("$nome possui ${nome.length} caracteres")
}
}
```

Objetos nulos (Null Safety)

- Ou então utilizar o operador de safe call (?)

- – Ignora a chamada se o objeto for nulo

```
fun main(){
var nome:String? = "Fabio"
println("Olá $nome")
nome = null // OK
println("Olá $nome")
println("$nome possui ${nome?.length} caracteres") // Erro de compilação
}
```

Para saber mais: https://kotlinlang.org/docs/reference/nullsafety.htm

Objetos nulos (Null Safety)

```
fun escreve(nome:String,  sNome:String?=null, faculdade:String="Impacta"){

if (sNome!=null){

 print(" $sNome")

}

if (faculdade!=null){

 print(" - $faculdade")

}

    println(" >>>") }

fun main(){

 escreve("José","Silva","GV")

  escreve("João","Silveira")

escreve("Ana")

escreve("Paula",faculdade="USP")

}
```

Funções – argumentos padrão

- Parâmetros de funções podem ter valores padrão, evitando a sobrecarga de métodos

```
fun main(args: Array<String>) {
var i = getInteiro("5")
println(i)
i = getInteiro(null)
println(i)
i = getInteiro(null, 2)
println(i)
}
// Função que transforma uma string num
      inteiro; caso a string seja nula,
// retorna 0, o valor doo argumento padrão
fun getInteiro(s: String?, padrao: Int = 0): Int
      {
if (s != null) {
return s.toInt()
}
return padrao
}
```

- Mais:https://kotlinlang.org/docs/reference/functions.html#defaultarguments

Funções – argumentos nomeados

- O nome do parâmetro pode ser utilizado na chamada da função
  - Possibilita passagem de parâmetros em qualquer ordem

```
fun main() {
teste("Fabio", "Pereira", "Impacta")
teste("Fabio")
teste("Fabio", faculdade = "Impacta")
}
fun teste(nome: String?, sobrenome: String? = null, faculdade: String?
= null) {
println("Nome: $nome, Sobrenome: $sobrenome, Faculdade:
$faculdade")
}
```

- Mais: https://kotlinlang.org/docs/reference/functions.html#named-arguments

Funções – varargs

- Uma função que tem um parâmetro varargs (normalmente o último) pode receber um ou mais parâmetros, separados por vírgula
  - Utiliza-se a palavra reservada vararg antes do parâmetro

```
fun main(args: Array<String>) {
var list = toList("ADS","BD","GTI")
print(list)
}
fun toList(vararg args: String): List<String>{
val list = ArrayList<String>()
for (s in args)
list.add(s)
return list
}
```

- Mais: https://kotlinlang.org/docs/reference/functions.html#variable-number-of-arguments-varargs

Exercícios

- 14) Informe dois valores em variáveis e um terceiro valor em uma String, indicando qual a operação a ser realizada. Na função valide se a operação trata-se de uma soma, subtração, multiplicação ou divisão e execute a operação
- 15) Modularize todos os exercícios do 1 ao 12, utilizando funções

Resolução – Exercício 14

```
fun calcula(num1:Double, num2:Double, operacao:String){
var resultado:Double = 0.0
if(operacao == "SOMAR"){
resultado = num1+num2}
else if (operacao == "SUBTRAIR"){
resultado = num1-num2
}else if (operacao=="MULTIPLICAR") {
resultado = num1*num2
}else if (operacao == "DIVIDIR"){
resultado = num1/num2}
print("$operacao, $resultado")
}
fun main(){
var num1:Double
var num2:Double
var operacao:String
num1 = 30.0
num2 = 20.0
operacao = "SUBTRAIR"
calcula(num1, num2, operacao)
}
```

String Templates

- Utilizar o valor de uma variável para imprimir ou com outra string sem necessidade de concatenar
  - $variavel ou ${objeto.propriedade}

```
fun main(args: Array<String>) {
var nome = "Fabio"
println("Olá $nome")
println("$nome possui ${nome.length}")
var nome_completo = "$nome Silva"
println(nome_completo)
}
```

Para saber mais: https://kotlinlang.org/docs/reference/basic-types.html#string-templates

VAR e VAL

- var: criar variável
- val: criar uma constante ou variável somente de leitura (valor atribuído não pode ser alterado)
- Sintaxe básica:
  - var variavel:tipo = valor
  - val variavel:tipo = valor
- O tipo vem depois do nome da variável, separado por :, e pode ser omitido caso seja feita uma atribuição na declaração
  - var variavel = valor
  - val variavel = valor

- Para saber mais: https://kotlinlang.org/docs/reference/basic-syntax.html

VAR e VAL - Exemplo

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

var nome:String = "Fabio"

println("Olá $nome")

nome = "Rodrigo"

println("Olá $nome")

}

fun main(args: Array<String>) {

var nome:String = " Fabio"

println("Olá $nome")

var sobrenome = "Pereira"

println("Olá $nome $sobrenome")

 }

fun main(args: Array<String>) {

val nome:String = "Fabio"

println("Olá $nome")

nome = "Pereira" // erro de compilação - Val cannot be reassigned println("Olá $nome")

 }
```

Conversão de tipos: as e is

- Conversão de tipos: as e is

- as? : cast seguro. Retorna null caso a conversão não possa ser feita (exceção)
  - is: verificar se uma variável é de um tipo
  - Se utilizado dentro de um if e for verdadeiro, a conversão é automática (Smart Cast)

```
fun main(args: Array<String>) {
var s:Any = "Fabio"
println(s as String) // transforma s em uma String
 println(s as? Int) // cast seguro: não é possível converter String em Int
if (s is String) { // verdadeiro: converte s em uma String
println("$s é uma string")
}}
```

Operador ternário e Elvis

- Feito com if/else

```
fun parOuImpar(a: Int): String {

return if (a % 2 == 0) "par" else "impar"

}

fun main(args: Array<String>) {

println(parOuImpar(1))

println(parOuImpar(2))

}
```

- Elvis (?:)
  - Se o valor da variável não for nulo, utilize seu valor; caso contrário, utilize outro

```
fun enviarEmail(usuario: String, titulo: String? = null): String {
val s = titulo?: "Bem vindo"
return "$s $usuario"
}
fun main(args: Array<String>) {
println(enviarEmail("Fabio"))
println(enviarEmail("Fabio", "Olá"))
}
```

Operador ternário e Elvis

- Para saber mais:
  - https://kotlinlang.org/docs/reference/control-_flow.html
  - https://kotlinlang.org/docs/reference/null-_safety.html

Referências

- https://developer.android.com/training/basics/firstapp?hl=pt-br
- https://www.sncticet.ufam.edu.br/2017/downloads/christianreis.pdf
- https://homepages.dcc.ufmg.br/~fernando/classes/android/slides/Class1.pdf
- https://pt.slideshare.net/AnaDoloresLimaDias/android-9149956
- https://developer.android.com/guide/components/activities/activity-lifecycle?hl=pt-br
- https://www.tutorialspoint.com/android/android_hello_world_example.htm
- https://kotlinlang.org/docs/home.html