Fundamentos do Desenvolvimento Android

Rafael Leal

Etapa 01

Sumário

Objetivos Gerais

1 $\rightarrow$ Desenvolver aplicativos com múltiplas telas usando Kotlin;

Objetivos da aula

Etapa 01: Introdução à Linguagem Kotlin

1 → Entender como programação procedural funciona

2 → Entender a estrutura e comandos básicos do Kotlin

Comece já

Palataforma de programação Kotlin on-line

Sintaxe Básica

Comentários podem ser

- entre /* ... */ para várias linhas;
- depois de // para linha única.

O programa inicia na função main

```
/**
 * Comentário multilinhas
 */

// Comentário de uma única linha

// Ponto de entrada: função main:
fun main() {

}
```

Referência

Sintaxe Básica

- A palavra chave **val** define um armazenamento local que **não pode** ser alterado.
- O tipo da variável vem depois do nome e começa com letra maiúscula
- O tipo pode ser induzido pelo valor inicializado. (b é do tipo Int)

```
fun main() {

    // Armazenamento de valores inteiros
    val a: Int = 1;
    val b = 2;
    val c = 3
}
```

Ponto e vírgula é opcional no final das linhas em Kotlin
Referência

Sintaxe Básica

- A palavra chave **var** define um armazenamento local que **pode** ser alterado.
- O tipo da variável vem depois do nome e começa com letra maiúscula
- O tipo pode ser induzido pelo valor inicializado. (y é do tipo Int)

```
fun main() {

    // Armazenamento de variáveis inteiras
    var x: Int = 1;
    var y = 2;
    var z = 3
}
```

Referência

Sintaxe Básica

- Os texto são armazenados como **String** e ficam entre **aspas duplas**.
- Caracteres são armazenados como **Char** e ficam entre **aspas simples**.
- Uma String é uma lista de itens do tipo Char.

```
fun main() {

    // Armazenamento de variáveis inteiras
    var texto: String = "Texto entre aspas duplas";
    var caractere: Char = 'c';
    var t1 = "Essa é uma string"
    var t2 = 'A' // Esse é um caractere

}
```

Referência - characters
Referência - strings

Sintaxe Básica

- A função print mostra na tela o texto que recebe como parâmetro.
- **print ("Texto mostrado na tela")**
- **println ("Texto mostrado na tela com quebra de linha ao final")**

```
fun main() {

    println("Texto com quebra de linha no final")
    print("Texto mostrado na tela")
    print("- Mais texto -")

}
```

```
Texto com quebra de linha no final
Texto mostrado na tela- Mais texto -
```

Referência

Sintaxe Básica

- Operação de soma entre Strings concatena o conteúdo
- Operação de soma entre Ints soma o conteúdo
- Operação de soma entre String e um Int concatena o conteúdo
- String + Int = String
  Int + String = Erro

```
fun main() {

    val umText = "1"
    val doisText = "2"
    var somaText = umText + doisText

    val dezInt = 10
    val vinteInt = 20
    var somaInt = dezInt + vinteInt

    var somaMisturada1 = umText + dezInt
    // Não é aceito soma de Int com String
    // var somaMisturada2 = vinteInt + doisText
    println(somaText)
    println(somaInt)
    println(somaMisturada1)

}
```

```
12
30
110
```

Sintaxe Básica

- Interpolação: o valor de **doisInt** é substituído em **$doisInt** no texto
- Interpolação: para valores mais complexos coloque ente chaves ${ 5*doisInt - 1 } faz a operação e depois substitui no texto
- Para inserir o caratere **$** use **${'$'}** no texto

```
fun main() {

    val doisInt = 2
    val texto1 = "Número " + doisInt + " é par"
    val texto2 = "Número $doisInt é par"
    val texto3 = "Valor: R${'$'} $doisInt,00"
    val texto4 = "5*doisInt - 1 = ${ 5*doisInt - 1 }"

    println(texto1) // Concatenação
    println(texto2) // interpolação
    println(texto3) // charactere $ inserido
    println(texto4) // interpolação com operação

}
```

```
Número 2 é par
Número 2 é par
Valor: R$ 2,00
5*doisInt - 1 = 9
```

Sintaxe Básica

- Operador condicional **if**:
  if (- condição -) - resultado se verdadeiro -
- Operador condicional **if** com várias linhas:
  if (- condição -) {
  - resultado linha 1-
  - resultado linha 2-
  }

```
fun main() {

    val idadeJoao = 18
    val idadePedro = 16

    if (idadeJoao > 17) println("Essa pessoa pode dirigir")
    if (idadePedro >= 16) println("Essa pessoa pode votar")
    if (idadeJoao == 18) println("Feliz aniversário")
    if (idadeJoao != 18) println("Essa linha não será mostrada")
    if (idadeJoao != 17){
        println("Teste diferente de 17")
        println("Teste com duas linhas de output use {}")
    }

}
```

```
Essa pessoa pode dirigir
Essa pessoa pode votar
Feliz aniversário
Teste diferente de 17
Teste com duas linhas de output use {}
```

Sintaxe Básica

Operadores condicionais **if - else if - else** :
if (- condição 1 -) {
- resultado se condição 1 for verdadeira -
} else if (- condição 2 -) {
- resultado se condição 1 for falsa, mas se a 2 for verdadeira -
} else {
- resultado se ambas forem falsas -
}

```
fun main() {

    val idadeJoao = 18
    val idadePedro = 16
    val idadeRafael = 12

    var idade = idadeJoao

    if (idade >= 18){
        println("Essa pessoa pode dirigir e votar.")
    } else if (idade >= 16) {
        println("Essa pessoa pode votar, mas NÃO dirigir.")
    } else {
        println("Essa pessoa NÃO pode votar NEM dirigir.")
    }

}
```

```
Essa pessoa pode dirigir e votar.
```

Sintaxe Básica

- Variáveis anuláveis recebem o sufixo ? no tipo

- var a: Int = null
  gera erro de compilação

```
fun main() {

    var idadeJoao: Int = 18 // variável não anulável
    var idadePedro: Int? = 16 // variável anulável
    val idadeRafael = 12

    var idade: Int? = null
    if (idade != null) {
        println("Idade: ${idade}")
    } else {
        println("Idade não informada")
    }

    idade = idadeRafael
    if (idade != null) {
        println("Idade: ${idade}")
    } else {
        println("Idade não informada")
    }
}
```

```
Idade não informada
Idade: 12
```

Sintaxe Básica

- Com Safe calls podemos usar propriedades de elementos anuláveis com segurança

```
fun main() {

    var texto : String? = null

    // Gera erro de compilação
    // println(texto.length)

    // retorna nulo, pois texto = null
    println(texto?.length)

    texto = "Alguma coisa"

    // retorna o tamanho do texto, pois texto != null
    println(texto?.length)
}
```

```
null
12
```

Sintaxe Básica

- O texto.let{ - comando - } só faz algo se o texto não for nulo.

```
fun main() {

    var texto : String? = null

    texto?.let{
        println("Teste 01: imprime se texto for não nulo")
    }

    texto = "Alguma coisa"

    texto?.let{
        println("Teste 02: imprime se texto for não nulo")
    }
}
```

```
Teste 02: imprime se texto for não nulo
```

Sintaxe Básica

- Com o operador Elvis podemos retornar um valor padrão se a variável for nula.

```
fun main() {

    val texto : String = "texto não nulo"

    var resposta: String? = null

    println(resposta?.length)
    println(resposta?.length ?: 0)
    resposta = texto
    println(resposta?.length ?: 0)

}
```

```
null
0
14
```

Desafios

Sabendo que texto.repeat(n) repete o texto n vezes faça um código que imprime a pirâmide:

1
22
333
4444

Sabendo que **texto.repeat(n)** repete o texto n vezes faça um código que imprime a pirâmide centralizada:

```
  1
 333
55555
```

Respostas

```
fun main() {

    println("1".repeat(1))
    println("2".repeat(2))
    println("3".repeat(3))
    println("4".repeat(4))

}
```

```
1
22
333
4444
```

```
fun main() {
    print(" ".repeat(2))
    println("1".repeat(1))
    print(" ".repeat(1))
    println("3".repeat(3))
    println("5".repeat(5))
}
```

```
  1
 333
55555
```

Sintaxe Básica

Usando funções podemos tornar o código mais simples e reaproveitável.
Para cada linha da primeira pirâmide podemos criar uma função com um valor de entrada e ela imprime a linha.
Usando a palavra chave **fun** e dentro dos parênteses dizendo o valor de entrada podemos repetir o processo a cada chamada.

```
fun linhaPiramide(num: Int){
    println("$num".repeat(num))
}

fun main() {
    linhaPiramide(1)
    linhaPiramide(2)
    linhaPiramide(3)
    linhaPiramide(4)
}
```

```
1
22
333
4444
```

Sintaxe Básica

Para o segundo exemplo podemos também dizer quantos espaços em branco queremos antes de cada print. Nesse caso separamos os valores de entrada por vírgulas.
Também é possível retornar um valor ao final da execução da função.

Desafio: some os valores das pirâmides

```
fun linhaPiramide(num: Int, espacos: Int){
    print(" ".repeat(espacos))
    println("$num".repeat(num))
}

fun main() {
    linhaPiramide(1, 2)
    linhaPiramide(3, 1)
    linhaPiramide(5, 0)
}
```

```
  1
 333
55555
```

Sintaxe Básica

A primeira função recebe dois valores de entrada e retorna a soma da linha. No meio do processo ela chama a segunda função que imprime a linha da pirâmide. Assim evita repetição de código e salva o valor de cada iteração.

```
fun somaLinhaPiramide(num: Int, espacos: Int) : Int{
    linhaPiramide(num, espacos)
    return num*num
}

fun LinhaPiramide(num: Int, espacos: Int){
    print(" ".repeat(espacos))
    println("$num".repeat(num))
}

fun main() {
    var soma = 0
    soma+= somaLinhaPiramide(1, 2)
    soma+= somaLinhaPiramide(3, 1)
    soma+= somaLinhaPiramide(5, 0)
    println("A soma dos valores da pirâmide é ${soma}")
}
```

```
  1
 333
55555
 A soma dos valores da pirâmide é 35
```

Sintaxe Básica

Podemos armazenar valores em lista também. Vamos agora salvar a soma de cada linha da pirâmide e depois somar tudo.
Uma MutableList<**Type**> é uma lista com itens do tipo **Type**. Ela é inicialmente declarada como um alista vazia usando a função mutableListOf<Type>()

```
fun somaLinhaPiramide(
    num: Int,
    espacos: Int) : Int{
    linhaPiramide(num, espacos)
    return num*num
}

fun linhaPiramide(num: Int, espacos: Int){
    print(" ".repeat(espacos))
    println("$num".repeat(num))
}

fun main() {
    var valorLinhas: MutableList<Int> = mutableListOf<Int>()
    valorLinhas.add( somaLinhaPiramide(1, 2)  )
    valorLinhas.add( somaLinhaPiramide(3, 1)  )
    valorLinhas.add( somaLinhaPiramide(5, 0)  )
    println("Os valores das linhas são: ${valorLinhas}")
    println("A soma total é: ${valorLinhas.sum()}.")
}
```

```
  1
 333
55555
Os valores das linhas são: [1, 9, 25]
A soma total é: 35
```

Sintaxe Básica

Para repetir um processo para valores de 1 a 3 usamos a estrutura de laço for
for (num **in** 1..3) {
-processo repetido para num = 1, 2, 3-
}
Nesse exemplo usa um passo de 2
for (num **in** 1..5 step 2) {
-processo repetido para num = 1, 3, 5-
}
Nesse exemplo conta de traz para frente
for (num **in** 3 downTo 1) {
-processo repetido para num = 3, 2, 1-
}

```
fun somaLinhaPiramide(
    num: Int,
    espacos: Int) : Int{
    linhaPiramide(num, espacos)
    return num*num
}

fun linhaPiramide(num: Int, espacos: Int){
    print(" ".repeat(espacos))
    println("$num".repeat(num))
}

fun main() {
    var valorLinhas: MutableList<Int> = mutableListOf<Int>()
    for (num in 1..3) {
        valorLinhas.add( somaLinhaPiramide(num, 3-num)  )
    }
    println("Os valores das linhas são: ${valorLinhas}")
    println("A soma total é: ${valorLinhas.sum()}.")
}
```

```
 1
 22
333
Os valores das linhas são: [1, 4, 9]
A soma total é: 14.
```

Desafios

1. Imprima a pirâmide de cabeça pra baixo:

   55555
    333
     1
2. Crie uma função que coloca a primeira letra de cada palavra maiúscula.
   Dica: texto.uppercase() torna o texto maiúsculo
3. Crie uma função que conte a quantidade de números pares de uma lista. Ex.:
   contaPar(listOf(2, 7, 8)) = 2
   Dica: se num%2 == 0 então num é par.
4. Crie uma função que formate o cpf Ex.:
   formataCpf("12345678910") = "123.456.789-10"

Respostas

Dentro do for faça a contagem de traz pra frente usando um passo de 2.
for (num **in** 5 downTo 1 step 2)
Além disso o numere de espaços tem que ser dividio por 2 pra centralizar a pirâmide.
somaLinhaPiramide(num, (**5**-num)/**2**)

```
fun somaLinhaPiramide(
    num: Int,
    espacos: Int) : Int{
    linhaPiramide(num, espacos)
    return num*num
}

fun linhaPiramide(num: Int, espacos: Int){
    print(" ".repeat(espacos))
    println("$num".repeat(num))
}

fun main() {
    var valorLinhas: MutableList<Int> = mutableListOf<Int>()
    for (num in 5 downTo 1 step 2) {
        valorLinhas.add( somaLinhaPiramide(num, (5-num)/2)  )
    }
    println("Os valores das linhas são: ${valorLinhas}")
    println("A soma total é: ${valorLinhas.sum()}.")
}
```

```
55555
 333
  1
Os valores das linhas são: [25, 9, 1]
A soma total é: 35.
```

Respostas

Os índices na lista começam a contar do zero. nome[0] é a primeira letra e deve ser maiúsculas. a primeira letra de cada sobrenome vem depois do espaço, entaõ devemos verificar se a letra anterior é ' ', se for então essa letra é maiúscula.

for (c **in** 1 until nome.length){

conta de 1 até o tamanho da palavra nome - 1.

Outra maneira de fazer esse laço é

for (c **in** 1..nome.length-1){

```
fun tornaMaiusculas(nome: String): String {
    var nomeNovo = ""
    nomeNovo+=nome[0].uppercase()
    for (c in 1 until nome.length){
        if(nome[c-1]==' '){
            nomeNovo+=nome[c].uppercase()
        } else {
            nomeNovo+=nome[c]
        }
    }
    return nomeNovo
}

fun main() {

    val nomeAntigo = "rafael leal"
    var nomeNovo = tornaMaiusculas("rafael leal")
    println("Nome antigo = $nomeAntigo")
    println("Novo nome   = $nomeNovo")
}
```

```
Nome antigo = rafael leal
Novo nome   = Rafael Leal
```

Respostas

Inicializamos o contador como zero e para cada vez que encontramos um número par na lista somamos um. ao final retornamos o contador.
for (num **in** lista){ percorre todos os itens da lista.
As seguintes expressões fazem a mesma coisa:

- contador ++
- contador += 1
- contador = contador +1

```
fun contaPares(lista: List<Int>): Int {
    var contador = 0
    for (num in lista){
        if (num % 2 == 0) {
            contador ++
        }
    }
    return contador
}

fun main() {

    val lista = listOf(1,2,3,4,79,9)
    println("A quantidade de números pares na lista")
    println("$lista ")
    println("é ${contaPares(lista)} ")

}
```

```
A quantidade de números pares na lista
[1, 2, 3, 4, 79, 9]
é 2
```

Respostas

A função texto.subString(a,b) copia a parte da string começa na posição a até a anterior à posição b.
var text: String = "texto "
posição 0: 't'
posição 1: 'e'
posição 2: 'x'
posição 3: 't'
posição 4: 'o'
posição 5: ' '
text.subString(2,4) = "xt"

```
fun formataCpf(cpf: String) : String {

    var cpfNovo = ""
    cpfNovo += cpf.substring(0,3)
    cpfNovo +="."
    cpfNovo += cpf.substring(3,6)
    cpfNovo +="."
    cpfNovo += cpf.substring(6,9)
    cpfNovo +="-"
    cpfNovo += cpf.substring(9,11)

    return cpfNovo
}

fun main() {
    val cpfAntigo = "12345678910"
    println("O CPF antigo é: $cpfAntigo")
    println("O CPF novo é  : ${formataCpf(cpfAntigo)}")

}
```

```
O CPF antigo é: 12345678910
O CPF novo é  : 123.456.789-10
```

Exercício

Crie uma função que retorne as vogais usadas na string e quantas são.

```
fun contaVogais(texto: String) : List<String> {

    var a = 0
    var e = 0
    var i = 0
    var o = 0
    var u = 0

    texto.forEach{ it ->
                if (it =='a') {a++}
                if (it =='e') {e++}
                if (it =='i') {i++}
                if (it =='o') {o++}
                if (it =='u') {u++}

    }
    return  listOf("a = $a", "e = $e", "i = $i", "o = $o", "u = $u")
}
```

```
fun main() {

    var nome = "Rafael Leal"
    println("Nome: ${nome}")
    println("${contaVogais(nome)}")

    var instituto = "Infnet"
    println("Instituto: ${instituto}")
    println("${contaVogais(instituto)}")

}
```

```
Nome: Rafael Leal
[a = 3, e = 2, i = 0, o = 0, u = 0]
Instituto: Infnet
[a = 0, e = 1, i = 0, o = 0, u = 0]
```

Qual erro deu no segundo caso?

Sintaxe Básica

Outra maneira de resolver o problema é usando o **when**. Quando it for igual a um termo 'esquerda da seta faça o que está à direita.

```
fun contaVogais2(texto: String) : List<String> {

    var a = 0
    var e = 0
    var i = 0
    var o = 0
    var u = 0

    texto.forEach{ it ->
        when(it){
            'a', 'A'-> a++
            'e', 'E'-> e++
            'i', 'I'-> i++
            'o', 'O'-> o++
            'u', 'U'-> u++
        }

    }
    return  listOf("a = $a", "e = $e", "i = $i", "o = $o", "u = $u")
}
```

```
fun main() {

    var nome = "Rafael Leal"
    println("Nome: ${nome}")
    println("${contaVogais2(nome)}")

    var instituto = "Infnet"
    println("Instituto: ${instituto}")
    println("${contaVogais2(instituto)}")

}
```

```
Nome: Rafael Leal
[a = 3, e = 2, i = 0, o = 0, u = 0]
Instituto: Infnet
[a = 0, e = 1, i = 1, o = 0, u = 0]
```

Agora as letras maiúsculas foram reconhecidas.

Referências

1. Palataforma de programação Kotlin on-line
2. Documentação Android
3. Conheça a linguagem de programação Kotlin
4. Site Oficial kotlin

Resumo

Vimos nessa aula:

→ Linguagem Kotlin

Competências: Desenvolver aplicativos com múltiplas telas usando Kotlin

Até a próxima!