

# Desenvolvimento de Aplicativos Móveis

Professor Maurício Buess

[mbuess@up.edu.br](mailto:mbuess@up.edu.br)



# Desenvolvimento Mobile

## Introdução ao Kotlin - (<https://play.kotlinlang.org/>)

- Declaração de constantes e variáveis:

`val`   `name` :   `data type`   =   `initial value`

- Exemplo:

`val _count: Int = 2;`

name   data type   initial value  
↓   ↓   ↙  
`val count: Int = 2`

## Introdução ao Kotlin

- Exemplo:

```
fun main() {  
    val _count: Int = 2;  
    println("Você tem $_count mensagens não lidas.");  
}
```

```
fun main() {  
    val unreadCount = 5;  
    val readCount = 100;  
    println("Você tem ${unreadCount+readCount} total de mensagem na caixa postal.");  
}
```

## Introdução ao Kotlin

```
fun main() {  
    val cartTotal = 0;  
    cartTotal = 20; // erro: Val cannot be reassigned  
    println("Total: $cartTotal");  
}
```

- **val** - Use quando você espera que o valor da variável não mude.
- **var** - Use quando você espera que o valor da variável possa mudar.

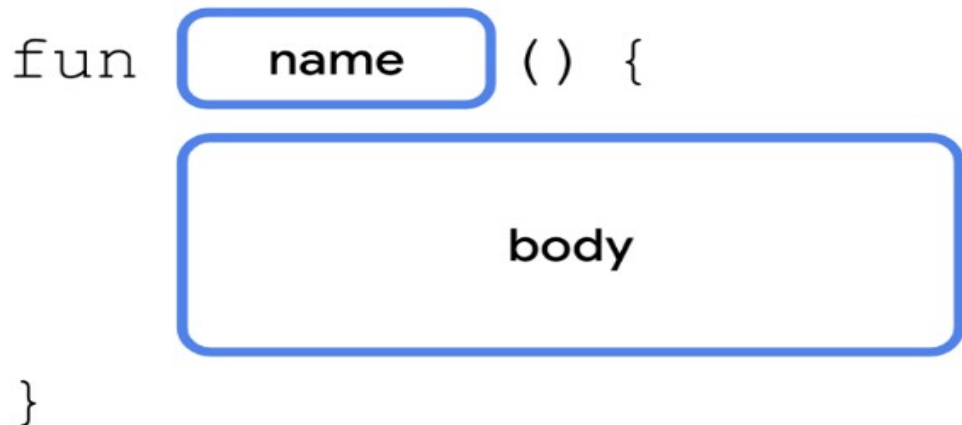
```
fun main() {  
    var cartTotal = 0;  
    cartTotal = 20;  
    println("Total: $cartTotal");  
}
```

# Desenvolvimento Mobile

## Criando funções em Kotlin

Função sem retorno  
são chamadas de  
Unit (void do Java)

```
fun main() {  
    birthdayGreeting();  
}  
fun birthdayGreeting() {  
    println("Parabéns, Asdrubal!");  
    println("Agora você tem 5 anos de idade!");  
}
```



The diagram illustrates the structure of a Kotlin function. It shows the keyword 'fun' followed by a function signature 'name' enclosed in a blue rounded rectangle, then the parentheses '()' and an opening curly brace '{'. Below this, a large blue rounded rectangle represents the function body, containing the word 'body'. The function ends with a closing curly brace '}'.

```
fun name () {  
    body  
}
```

## Criando funções - Retornar uma String

```
fun birthdayGreeting(): String {  
    val nameGreeting = "Feliz aniversário, Asdrubal!";  
    val ageGreeting = "Agora você tem 5 anos de idade!";  
    var retorno : String = nameGreeting + "\n" + ageGreeting;  
    return retorno;  
}
```

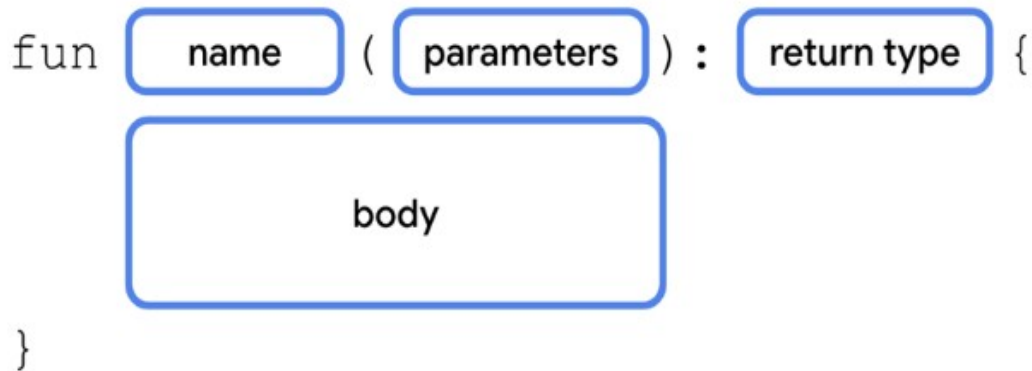
```
fun main() {  
    println(birthdayGreeting());  
}
```

# Desenvolvimento Mobile

## Criando funções – Parâmetro(s) de Entrada

```
fun birthdayGreeting(_nome : String) : String {  
    var nameGreeting : String  
    val ageGreeting = "Agora você tem 5 anos de idade!"  
    nameGreeting = "Feliz aniversário, " + _nome + "!"  
    var retorno : String = nameGreeting + "\n" + ageGreeting;  
    return retorno  
}
```

```
fun main() {  
    println(birthdayGreeting("Asdrubal"))  
}
```



# Desenvolvimento Mobile

## Criando funções – Parâmetro(s) de Entrada

```
fun birthdayGreeting(_nome : String, _idade : Int) : String {  
    var nameGreeting : String;  
    var ageGreeting = "Agora você tem $_idade anos de idade!";  
    nameGreeting = "Feliz aniversário, " + _nome + "!";  
    var retorno : String = nameGreeting + "\n" + ageGreeting;  
    return retorno;  
}
```

```
fun main() {  
    println(birthdayGreeting("Asdrubal", 15));  
    println(birthdayGreeting("Ágatha", 24));  
}
```



# Desenvolvimento Mobile

## Criando funções – Parâmetro(s) de Entrada

```
fun birthdayGreeting(_nome : String, _idade : Int) : String {  
    var nameGreeting : String;  
    var ageGreeting = "Agora você tem $_idade anos de idade!";  
    nameGreeting = "Feliz aniversário, " + _nome + "!";  
    var retorno : String = nameGreeting + "\n" + ageGreeting;  
    return retorno;  
}  
fun main() {  
    var _nivers : String;  
    _nivers = birthdayGreeting("Asdrubal", 15) + "\n\n" + birthdayGreeting("Ágatha", 24);  
    println(_nivers);  
}
```

## Criando funções – Resumo

### Resumo

- As funções são definidas usando a palavra-chave `fun` e contém partes de código reutilizáveis.
- As funções facilitam a manutenção de programas maiores e evitam a repetição desnecessária de código.
- As funções podem retornar um valor que pode ser armazenado em uma variável para uso futuro.
- As funções podem ter parâmetros, que são variáveis disponíveis no corpo da função.
- Argumentos são os valores que você transmite quando chama uma função.
- Você pode nomear argumentos ao chamar uma função. Ao usar argumentos nomeados, é possível reordenar os argumentos sem afetar a saída.
- É possível especificar um argumento padrão para omitir o argumento ao chamar uma função.

## Criando funções – Estruturas de desvio

```
fun main() {  
    val a = 9;  
    val b = 10;  
  
    var _max = a;  
  
    if(b > a) {  
        _max = b;  
    }  
  
    println(_max);  
}
```

## Criando funções – Estruturas de desvio

```
fun main() {  
    val a = 9;  
    val b = 10;  
    var _max = a;  
    if(b < a) {  
        _max = b;  
    } else {  
        _max = 0;  
    }  
    println(_max);  
}
```

## Criando funções – Estruturas de desvio

```
fun main() {  
    val estado = "ativo";  
  
    when(estado) {  
        "ativo" -> print("Clique para desativar");  
        "inativo" -> print("Clique para ativar");  
    }  
}
```

## Criando funções – Estruturas de desvio

```
fun main() {  
    val estado = "suspenso";  
  
    when(estado) {  
        "ativo" -> print("Clique para desativar");  
        "inativo" -> print("Clique para ativar");  
        else -> {  
            print("valor de estado é desconhecido")  
        }  
    }  
}
```

## Criando funções – Estruturas de desvio

```
fun main() {  
    val poltrona = 2  
  
    when(poltrona) {  
        in 1..10 -> println("Fileira A")  
        in 11..20 -> println("Fileira B")  
        in 21..30 -> println("Fileira C")  
        in 31..40 -> println("Fileira D")  
    }  
}
```

# Desenvolvimento Mobile

## Criando funções – Estruturas de repetição

```
fun main() {  
    val colecao = 1..9  
  
    for (item in colecao) {  
        print(item)  
    }  
}
```



# Desenvolvimento Mobile

## Criando funções – Estruturas de repetição

```
fun main() {  
    val colecao = 9;  
    var i = 0  
  
    while (i <= colecao) {  
        print(i);  
        i++;  
    }  
  
}
```

# Desenvolvimento Mobile

```
fun updateWeather(degrees:Int){  
    val descricao:String  
    val cor:String  
    if(degrees<10){  
        descricao = "Frio"  
        cor = "BLUE"  
    } else if (degrees<25){  
        descricao = "Ameno"  
        cor = "ORANGE"  
    } else {  
        descricao = "Calor"  
        cor = "RED"  
    }  
}
```

```
println("O clima está $descricao")  
println(cor)  
}
```

---

```
fun main() {  
    print(updateWeather(21))  
}
```

# Desenvolvimento Mobile

```
fun updateWeather(degrees:Int){  
    val (descricao:String, cor:String) =  
    if(degrees<10){  
        Pair("Frio", "BLUE")  
    } else if (degrees<25){  
        Pair("Ameno", "ORANGE")  
    } else {  
        Pair("Calor", "RED")  
    }  
    println("O clima está $descricao")  
    println(cor)  
}
```

```
fun main() {  
    print(updateWeather(21))  
}
```

# Desenvolvimento Mobile

```
fun updateWeather(degrees:Int){
    val (descricao, cor) = when {
        degrees<10 -> Pair("Frio", "BLUE")
        degrees<25 -> Pair("Ameno", "ORANGE")
        else -> Pair("Calor", "RED")
    }
    println("O clima está $descricao")
    println(cor)
}
fun main() {
    print(updateWeather(21))
}
```

```
fun main() {
    print(updateWeather(21))
}
```

# Desenvolvimento Mobile

```
fun updateWeather(degrees:Int){
    val (descricao, cor) = when {
        degrees<10 -> "Frio" to "BLUE"
        degrees<25 -> "Ameno" to "ORANGE"
        else -> "Calor" to "RED"
    }
    println("O clima está $descricao")
    println(cor)
}
fun main() {
    print(updateWeather(21))
}
```

```
fun main() {
    print(updateWeather(21))
}
```

# Desenvolvimento Mobile

```
fun XXoo() : String {  
    println("Calculando XXoo")  
    return "XXoo"  
}  
  
fun main() {  
    println("Primeiro ${XXoo()} Segundo ${XXoo()}")  
}
```

- Qual o resultado?

(a)

Calculando XXoo

Calculando XXoo

Primeiro XXoo Segundo Xxoo

(b)

Calculando XXoo

Primeiro XXoo Segundo Xxoo

# Desenvolvimento Mobile

```
fun imprimirNome(nomes : List<String>) {  
    for (nome in nomes) {  
        println(nome)  
    }  
}
```

```
fun main(){  
    val nomes = listOf("Ana", "Bruno", "Carlos", "Bia")  
    imprimirNome(nomes)  
}
```

# Desenvolvimento Mobile

## Exercício 01:

Escreva uma função em Kotlin chamada `maiorDeDois` que receba dois números inteiros como argumentos e imprima qual é o maior número. Use uma estrutura de desvio simples (if/else).



# Desenvolvimento Mobile

```
fun maiorDeDois(a: Int, b: Int) {  
    if (a > b) {  
        println("$a é maior que $b")  
    } else if (b > a) {  
        println("$b é maior que $a")  
    } else {  
        println("$a e $b são iguais")  
    }  
}  
  
fun main() {  
    maiorDeDois(5, 3)  
    maiorDeDois(2, 8)  
    maiorDeDois(4, 4)  
}
```

# Desenvolvimento Mobile

## Exercício 2:

Escreva uma função em Kotlin chamada `imprimirNumeros` que receba um número inteiro `n` e imprima todos os números de 1 até `n`.

# Desenvolvimento Mobile

```
fun imprimirNumeros(n: Int) {  
    for (i in 1..n) {  
        println(i)  
    }  
}
```

```
fun main() {  
    imprimirNumeros(5)  
    imprimirNumeros(10)  
}
```

# Desenvolvimento Mobile

## Exercício 3:

Escreva uma função em Kotlin chamada fatorial que receba um número inteiro  $n$  e retorne o fatorial de  $n$  ( $n!$ ). Use um laço de repetição (for).

# Desenvolvimento Mobile

```
fun fatorial(n: Int): Long {  
    var resultado = 1L  
    for (i in 1..n) {  
        resultado *= i  
    }  
    return resultado  
}
```

```
fun main() {  
    println("Fatorial de 5: ${fatorial(5)}") // 120  
    println("Fatorial de 10: ${fatorial(10)}") // 3628800  
}
```

# Desenvolvimento Mobile

## Exercício 4:

Escreva uma função em Kotlin chamada `classificarIdade` que receba uma idade como argumento e imprima a classificação da idade de acordo com a tabela abaixo.

0 a 12: Criança

13 a 17: Adolescente

18 a 64: Adulto

65 ou mais: Idoso

# Desenvolvimento Mobile

```
fun classificarIdade(idade: Int) {  
    when (idade) {  
        in 0..12 -> println("Criança")  
        in 13..17 -> println("Adolescente")  
        in 18..64 -> println("Adulto")  
        in 65..Int.MAX_VALUE -> println("Idoso")  
        else -> println("Idade inválida")  
    }  
}
```

```
fun main() {  
    classificarIdade(8) // Criança  
    classificarIdade(15) //Adolescente  
    classificarIdade(30) //Adulto  
    classificarIdade(70) //Idoso  
    classificarIdade(-5) // Idade inválida  
}
```

# Desenvolvimento Mobile

## Exercício 5:

Escreva uma função em Kotlin chamada `somarLista` que receba uma lista de números inteiros e retorne a soma de todos os elementos da lista.



# Desenvolvimento Mobile

```
fun somarLista(numeros: List<Int>): Int {  
    var soma = 0  
    for (numero in numeros) {  
        soma += numero  
    }  
    return soma  
}  
fun main() {  
    val lista1 = listOf(1, 2, 3, 4, 5)  
    val lista2 = listOf(10, 20, 30, 40)  
  
    println("Soma da lista1: ${somarLista(lista1)}") // Deve imprimir: 15  
    println("Soma da lista2: ${somarLista(lista2)}") // Deve imprimir: 100  
}
```