

# REFERÊNCIA RÁPIDA DE KOTLIN

Segurança de nulos, coroutines, data classes, programação funcional

## Básico

```
Hello World
fun main() {
    println("Hello, World!")
}
```

## Variáveis

```
val name = "Kotlin" // immutable (prefer)
var count = 9 // mutable
val pi: Double = 3.14159 // explicit type
const val MAX = 100 // compile-time constant
```

## Tipos Básicos

**Int**, **Long** Inteiros com sinal de 32/64 bits  
**Double**, **Float** Ponto flutuante de 64/32 bits  
**Boolean** true / false  
**Char** Único caractere Unicode  
**String** Texto imutável, suporta templates  
**Unit** Equivalente a "void" (valor único)  
**Nothing** Função que nunca retorna (ex.: lança exceção)

## Templates de String

```
val name = "World"
println("Hello, $name!")
println("Length: ${name.length}")
val raw = """Line 1
|Line 2"""
```

## Funções

### Declaração de Função

```
fun add(a: Int, b: Int): Int {
    return a + b
}
fun add(a: Int, b: Int) = a + b // single expression
```

### Argumentos Padrão e Nomeados

```
fun greet(name: String, greeting: String = "Hello") {
    println("$greeting, $name!")
}
greet("Alice") // Hello, Alice!
greet("Bob", greeting = "Hi") // Hi, Bob!
```

### Funções de Ordem Superior

```
fun operate(a: Int, b: Int, op: (Int, Int) -> Int): Int {
    return op(a, b)
}
val sum = operate(3, 4) { a, b -> a + b }
```

### Varargs

```
fun sum(vararg nums: Int): Int = nums.sum()
sum(1, 2, 3)
val arr = intArrayOf(1, 2, 3)
sum(*arr) // spread operator
```

## Classes

### Definição de Classe

```
class Person(val name: String, var age: Int) {
    fun greet() = "Hi, I'm $name"
}
val p = Person("Alice", 30)
println(p.name)
```

### Herança

```
open class Shape(val sides: Int) { open fun area(): Double = 0.0 }
class Circle(val r: Double): Shape(0) {
    override fun area() = Math.PI * r * r
}
```

### Modificadores de Visibilidade

**public** Visível em qualquer lugar (padrão)  
**private** Visível dentro da classe / arquivo  
**protected** Classe e subclasses  
**internal** Somente no mesmo módulo

### Abstract e Interfaces

```
interface Drawable { fun draw() }
abstract class Widget : Drawable { abstract val label: String }
class Button(override val label: String): Widget() {
    override fun draw() = println("Drawing $label")
}
```

## Segurança de Nulos

### Tipos Anuláveis

```
var name: String? = null // nullable
val len = name?.length // safe call: null
val len2 = name?.length ?: 0 // Elvis operator: 0
val len3 = name!!.length // assert non-null (throws)
```

### Operações Seguras

**?.** Chamada segura — retorna null se receptor for null  
**?:** Elvis — valor padrão quando null  
**!!** Asserção não-null (lança se null)  
**? let { }** Executa bloco apenas se não-null  
**as?** Conversão segura — retorna null em falha

### Smart Casts

```
if (obj is String) println(obj.length) // auto-cast
when (obj) {
    is Int -> println(obj + 1)
    is String -> println(obj.uppercase())
}
```

## Coleções

### Criando Coleções

```
val list = listOf(1, 2, 3) // immutable
val mList = mutableListOf(1, 2, 3) // mutable
val map = mapOf("a" to 1, "b" to 2)
val set = setOf("x", "y", "z")
```

### Operações em Coleções

```
val nums = listOf(1, 2, 3, 4, 5)
nums.filter { it > 2 } // [3, 4, 5]
nums.map { it * 2 } // [2, 4, 6, 8, 10]
nums.firstOrNull { it > 3 } // 4
nums.sumOf { it } // 15
```

## Operações Comuns

**.filter { }** Mantém elementos que satisfazem o predicado  
**.map { }** Transforma cada elemento  
**.flatMap { }** Mapeia e achata  
**.groupBy { }** Agrupa por chave em Map  
**.sortedBy { }** Ordena por seletor  
**.associate { }** Transforma em Map (pares chave-valor)  
**.any { }** / **.all { }** Verifica se algum/todos satisfazem o predicado  
**.fold(initial) { }** Reduz com acumulador inicial

## Coroutines

### Coroutine Básica

```
import kotlinx.coroutines.*
fun main() = runBlocking {
    launch { delay(1000); println("World") }
    println("Hello")
}
```

### Async / Await

```
val deferred = async { fetchData() }
val result = deferred.await()
// parallel: launch multiple async, await all
val (a, b) = awaitAll(async { fetchA() }, async { fetchB() })
```

### Construtores de Coroutine

**launch { }** Coroutine fire-and-forget (retorna Job)  
**async { }** Retorna Deferred<T> com resultado  
**runBlocking { }** Conecta código bloqueante e suspenso

**withContext(dispatcher)** Troca o contexto da coroutine  
**coroutineScope { }** Escopo de concorrência estruturada

### Dispatchers

**Dispatchers.Default** Trabalho intensivo de CPU (pool de threads)  
**Dispatchers.IO** Operações de I/O bloqueantes  
**Dispatchers.Main** Thread principal/UI (Android, Swing)  
**Dispatchers.Unconfined** Inicia na thread do chamador, retoma em qualquer uma

## Extensões

### Funções de Extensão

```
fun String.isPalindrome(): Boolean {
    return this == this.reversed()
}
println("racecar".isPalindrome()) // true
```

### Propriedades de Extensão

```
val String.wordCount: Int
get() = this.split("\\s+").toRegex().size
println("hello world".wordCount) // 2
```

### Sobrecarga de Operador

```
data class Vec(val x: Double, val y: Double) {
    operator fun plus(other: Vec) = Vec(x + other.x, y + other.y)
}
val v = Vec(1.0, 2.0) + Vec(3.0, 4.0) // Vec(4.0, 6.0)
```

## Data Classes

### Data Class

```
data class User(val name: String, val age: Int)
val u1 = User("Alice", 30)
val u2 = u1.copy(age = 31) // non-destructive copy
val (name, age) = u1 // destructuring
```

### Membros Gerados Automaticamente

**equals()** Igualdade estrutural baseada nas propriedades  
**hashCode()** Consistente com equals()  
**toString()** User(name=Alice, age=30)  
**copy()** Cria cópia modificada  
**componentN()** Suporte a desestruturação

### Enum Classes

```
enum class Direction { NORTH, SOUTH, EAST, WEST }
val dir = Direction.NORTH
when (dir) { Direction.NORTH -> "up"; else -> "other" }
```

## Sealed Classes

### Hierarquia de Sealed Class

```
sealed class Result-out T {
    data class Success<T>(val data: T) : Result<T>()
    data class Error(val message: String) : Result<Nothing>()
    data object Loading : Result<Nothing>()
}
```

### When Exaustivo

```
fun handle(result: Result<String>): String = when (result) {
    is Result.Success -> result.data
    is Result.Error -> "Error: ${result.message}"
    is Result.Loading -> "Loading..."
} // no else needed - compiler checks exhaustiveness
```

### Sealed vs Enum

**Sealed class** Subclasses podem ter estado diferente  
**Sealed interface** Permite herança múltipla  
**Enum class** Conjunto fixo de instâncias singleton  
**data object** Singleton com sobrecrita de toString()

## Funções de Escopo

### Comparação de Funções de Escopo

**let** Contexto como `it`, retorna resultado do lambda  
**run** Contexto como `this`, retorna resultado do lambda

**with(obj)** Contexto como `this`, retorna resultado do lambda  
**apply** Contexto como `this`, retorna objeto de contexto  
**also** Contexto como `it`, retorna objeto de contexto

## let e apply

```
val name: String? = "Alice"
name?.let { println("Name is $it") }
val person = Person("Bob", 25).apply {
    age = 26 // configure object
}
```

## run e with

```
val result = "Hello".run { uppercase() + " WORLD" }
val info = with(person) { "$name is $age years old" }
```

## also

```
val numbers = mutableListOf(1, 2, 3)
.also { println("Original: $it") }
// also is useful for side effects (logging, validation)
```