

REFERENCIA RÁPIDA DE KOTLIN

Null safety, corrutinas, data classes, programación funcional

Fundamentos

Hola Mundo

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

Variables

```
val name = "Kotlin" // immutable (prefer)
var count = 0 // mutable
val pi: Double = 3.14159 // explicit type
const val MAX = 100 // compile-time constant
```

Tipos básicos

Int, **Long** Enteros con signo de 32/64 bits
Double, **Float** Punto flotante de 64/32 bits
Boolean true / false
Char Carácter Unicode individual
String Texto inmutable, admite plantillas
Unit Equivalente a `void` (un solo valor)
Nothing La función nunca retorna (p. ej., lanza excepción)

Plantillas de cadena

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

Funciones

Declaración de función

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

Argumentos por defecto y con nombre

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

Funciones de orden 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
```

Clases

Definición de clase

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

Herencia

```
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 visibilidad

public Visible en todos lados (por defecto)
private Visible dentro de la clase / archivo
protected Clase y subclases
internal Solo dentro del mismo módulo

Abstractas 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")
}
```

Null Safety

Tipos anulables

```
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)
```

Operaciones seguras

?. Llamada segura — retorna null si el receptor es null
?: Elvis — valor por defecto cuando es null
!! Aserción no-nula (lanza si es null)
? let { } Ejecutar bloque solo si no es null
as? Cast seguro — retorna null si falla

Smart casts

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

Colecciones

Crear colecciones

```
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")
```

Operaciones de colección

```
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
```

Operaciones comunes

.filter { } Conservar elementos que cumplen el predicado
.map { } Transformar cada elemento
.flatMap { } Mapear y aplanar
.groupBy { } Agrupar por clave en un Map
.sortedBy { } Ordenar por selector
.associate { } Transformar a Map (pares clave-valor)
.any { } / .all { } Verificar si alguno/todos cumplen el predicado
.fold(init) { } Reducir con acumulador inicial

Corrutinas

Corrutina 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() })
```

Constructores de corrutina

.launch { } Corrutina fire-and-forget (retorna Job)
async { } Retorna Deferred<T> con resultado
runBlocking { } Puente entre código bloqueante y suspendido
withContext(dispatcher) Cambiar el contexto de la corrutina
coroutineScope { } Scope de concurrencia estructurada

Dispatchers

Dispatchers.Default Trabajo intensivo de CPU (thread pool)
Dispatchers.IO Operaciones de E/S bloqueantes
Dispatchers.Main Hilo principal/UI (Android, Swing)
Dispatchers.Unconfined Inicia en el hilo del llamador, reanuda en cualquiera

Extensiones

Funciones de extensión

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

Propiedades de extensión

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

Sobrecarga de operadores

```
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
```

Miembros generados automáticamente

equals() Igualdad estructural basada en propiedades
hashCode() Consistente con `equals()`
toString() `User(name=Alice, age=30)`
copy() Crear copia modificada
componentN() Soporte para desestructuración

Clases enum

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

Clases selladas

Jerarquía de clase sellada

```
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 exhaustivo

```
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 Las subclases pueden tener estado diferente
Sealed interface Permite herencia múltiple
Enum class Conjunto fijo de instancias singleton
data object Singleton con override de `toString()`

Funciones de scope

Comparación de funciones de scope

let Contexto como `it`, retorna resultado del lambda
run Contexto como `this`, retorna resultado del lambda
with(obj) Contexto como `this`, retorna resultado del lambda

apply Contexto como `this`, retorna el objeto de contexto
also Contexto como `it`, retorna el objeto de contexto

let y apply

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

```

run y 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)
```