

Tham Khảo Nhanh Kotlin

Null safety, coroutines, data classes, lập trình hàm cơ bản

Cơ Bản

Hello World

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

Biến

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

Kiểu Cơ Bản

Int, Long	Số nguyên có dấu 32-bit / 64-bit
Double, Float	Số thực 64-bit / 32-bit
Boolean	true / false
Char	Ký tự Unicode đơn
String	Văn bản bất biến, hỗ trợ template
Unit	Tương đương void (giá trị đơn)
Nothing	Hàm không bao giờ trả về (vd: throws)

String Templates

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

Functions

Khai Báo Hàm

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

Đối Số Mặc Định & Có Tên

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

Higher-Order Functions

```
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

Định Nghĩa Class

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

Kế Thừa

```
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
}
```

Access Modifiers

public	Hiển thị mọi nơi (mặc định)
private	Hiển thị trong class / file
protected	Class và subclass
internal	Chỉ cùng module

Abstract & 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

Kiểu Nullable

```
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)
```

Thao Tác An Toàn

?	Safe call — trả về null nếu receiver là null
?:	Elvis — giá trị mặc định khi null
!!	Khẳng định không null (throws nếu null)
?..let { }	Thực thi block chỉ khi không null
as?	Safe cast — trả về null khi thất bại

Smart Casts

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

Collections

Tạo Collections

```
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")
```

Thao Tác Collection

```
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
```

Thao Tác Phổ Biến

.filter { }	Giữ phần tử khớp predicate
.map { }	Biến đổi từng phần tử
.flatMap { }	Map và làm phẳng
.groupBy { }	Nhóm theo key vào Map
.sortedBy { }	Sắp xếp theo selector
.associate { }	Biến đổi thành Map (cặp key-value)
.any { } / .all { }	Kiểm tra bất kỳ/tất cả khớp predicate
.fold(initial) { }	Rút gọn với accumulator ban đầu

Coroutines

Coroutine Cơ Bản

```
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() })
```

Coroutine Builders

launch { }	Fire-and-forget coroutine (trả về Job)
async { }	Trả về Deferred<T> với kết quả
runBlocking { }	Câu nối code blocking và suspending
withContext(dispatcher)	Chuyển đổi context coroutine
coroutineScope { }	Phạm vi concurrency có cấu trúc

Dispatchers

Dispatchers.Default	Công việc tốn CPU (thread pool)
Dispatchers.IO	Thao tác I/O blocking
Dispatchers.Main	Luồng Main/UI (Android, Swing)
Dispatchers.Unconfined	Bắt đầu trong luồng gọi, tiếp tục ở bất kỳ đâu

Extensions

Extension Functions

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

Extension Properties

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

Nạp Chồng Toán Tử

```
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
```

Thành Viên Tự Động Tạo

equals()	Bằng cấu trúc dựa trên thuộc tính
hashCode()	Nhất quán với equals()
toString()	User(name=Alice, age=30)
copy()	Tạo bản sao được sửa đổi
componentN()	Hỗ trợ destructuring

Tham Khảo Nhanh Kotlin

Enum Classes

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

Sealed Classes

Phân Cấp 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>()
}
```

Exhaustive When

```
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	Subclass có thể giữ trạng thái khác nhau
Sealed interface	Cho phép đa kế thừa
Enum class	Tập cố định các instance singleton
data object	Singleton với override toString()

Scope Functions

So Sánh Scope Functions

let	Context là it , trả về kết quả lambda
run	Context là this , trả về kết quả lambda
with(obj)	Context là this , trả về kết quả lambda
apply	Context là this , trả về context object
also	Context là it , trả về context object

let & apply

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

run & 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 { it.add(4) }
// also is useful for side effects (logging, validation)
```