

Referensi Cepat Kotlin

Null safety, coroutine, data class, pemrograman fungsional

Dasar

Hello World

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

Variabel

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

Tipe Dasar

Int, Long	Integer bertanda 32-bit / 64-bit
Double, Float	Bilangan pecahan 64-bit / 32-bit
Boolean	true / false
Char	Karakter Unicode tunggal
String	Teks immutable, mendukung template
Unit	Setara dengan void (nilai tunggal)
Nothing	Fungsi tidak pernah kembali (mis. throws)

Template String

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

Fungsi

Deklarasi Fungsi

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

Argumen Default & Bernama

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

Higher-Order Function

```
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
```

Class

Definisi Class

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

Pewarisan

```
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
}
```

Modifier Visibilitas

public	Terlihat di mana saja (default)
private	Terlihat dalam class / file
protected	Class dan subclass
internal	Modul yang sama saja

Abstract & Interface

```
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

Tipe 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)
```

Operasi Aman

?.	Safe call — mengembalikan null jika receiver null
?:	Elvis — nilai default ketika null
!!	Assertion non-null (throws jika null)
?..let { }	Jalankan blok hanya jika non-null
as?	Safe cast — mengembalikan null jika gagal

Smart Cast

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

Collection

Membuat Collection

```
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")
```

Operasi 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
```

Operasi Umum

.filter { }	Simpan elemen yang cocok dengan predicate
.map { }	Transformasi setiap elemen
.flatMap { }	Map dan ratakan
.groupBy { }	Kelompokkan berdasarkan key menjadi Map
.sortedBy { }	Urutkan berdasarkan selector
.associate { }	Transformasi ke Map (pasangan key-value)
.any { } / .all { }	Periksa apakah ada/semua yang cocok dengan predicate
.fold(init) { }	Reduce dengan akumulator awal

Coroutine

Coroutine Dasar

```
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() })
```

Builder Coroutine

launch { }	Fire-and-forget coroutine (mengembalikan Job)
async { }	Mengembalikan Deferred<T> dengan hasil
runBlocking { }	Menjembatani kode blocking dan suspending
withContext(dispatcher)	Ganti konteks coroutine
coroutineScope { }	Lingkup structured concurrency

Dispatcher

Dispatchers.Default	Pekerjaan CPU-intensive (thread pool)
Dispatchers.IO	Operasi I/O blocking
Dispatchers.Main	Thread main/UI (Android, Swing)
Dispatchers.Unconfined	Mulai di thread pemanggil, lanjut di thread mana pun

Extension

Extension Function

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

Extension Property

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

Operator Overloading

```
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)
```

Referensi Cepat Kotlin

Data Class

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
```

Anggota yang Di-generate Otomatis

equals()	Kesamaan struktural berdasarkan properti
hashCode()	Konsisten dengan equals()
toString()	User(name=Alice, age=30)
copy()	Buat salinan yang dimodifikasi
componentN()	Dukungan destructuring

Enum Class

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

Sealed Class

Hierarki 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 Exhaustive

```
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 bisa menyimpan state berbeda
Sealed interface	Mengizinkan multiple inheritance
Enum class	Kumpulan tetap dari singleton
data object	Singleton dengan override toString()

Scope Function

Perbandingan Scope Function

let	Konteks sebagai it , mengembalikan hasil lambda
run	Konteks sebagai this , mengembalikan hasil lambda
with(obj)	Konteks sebagai this , mengembalikan hasil lambda
apply	Konteks sebagai this , mengembalikan objek konteks
also	Konteks sebagai it , mengembalikan objek konteks

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)
```