

RÉFÉRENCE RAPIDE KOTLIN

Sécurité null, coroutines, data classes, programmation fonctionnelle essentiels

Bases

Hello World

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

Variables

```
val name = "Kotlin" // immuable (à préférer)
var count = 9 // mutable
val pi: Double = 3.14159 // type explicite
const val MAX = 100 // constante à la compilation
```

Types de base

Int, Long Entiers signés 32 bits / 64 bits
Double, Float Virgule flottante 64 bits / 32 bits
Boolean true / false
Char Caractère Unicode unique
String Texte immuable, supporte les templates
Unit Equivalent de `void` (valeur unique)
Nothing La fonction ne retourne jamais (ex. lève une exception)

Templates de chaînes

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

Fonctions

Déclaration de fonction

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

Arguments par défaut et nommés

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

Fonctions d'ordre supérieur

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

Vargarg

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

Classes

Définition de classe

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

Héritage

```
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
}
```

Modificateurs de visibilité

public Visible partout (par défaut)
private Visible dans la classe / le fichier
protected Classe et sous-classes
internal Même module uniquement

Classes abstraites et 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")
}
```

Sécurité null

Types nullables

```
var name: String? = null // nullable
val len = name?.length // appel sécurisé : null
val len2 = name?.length ?: 0 // opérateur Elvis : 0
val len3 = name?.length // assertion non-null (lève exception)
```

Opérations sécurisées

?.. Appel sécurisé — retourne null si le récepteur est null
?: Elvis — valeur par défaut si null
!! Assertion non-null (lève une exception si null)
?..let { } Exécuter le bloc uniquement si non-null
as? Cast sécurisé — retourne null en cas d'échec

Casts intelligents

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

Collections

Créer des collections

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

Opérations sur les collections

```
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
```

Opérations courantes

.filter { } Garder les éléments correspondant au prédicat
.map { } Transformer chaque élément
.flatMap { } Mapper et aplattr
.groupBy { } Regrouper par clé dans une Map
.sortedBy { } Trier par sélecteur
.associate { } Transformer en Map (paires clé-valeur)
.any { } / .all { } Vérifier si quelq'un/tous correspondent au prédicat
.fold(initial) { } Réduire avec un accumulateur initial

Coroutines

Coroutine de base

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

Async / Await

```
val deferred = async { fetchData() }
val result = deferred.await()
// parallèle : lancer plusieurs async, attendre tous
val (a, b) = awaitAll(async { fetchA() }, async { fetchB() })
```

Constructeurs de coroutines

launch { } Coroutine fire-and-forget (retourne Job)
async { } Retourne Deferred<T> avec résultat
runBlocking { } Fait le pont entre code bloquant et suspendu

withContext(dispatcher) Changer le contexte de la coroutine
coroutineScope { } Portée de concurrence structurée

Dispatchers

Dispatchers.Default Travail intensif CPU (pool de threads)
Dispatchers.IO Opérations I/O bloquantes
Dispatchers.Main Thread principal/UI (Android, Swing)
Dispatchers.Unconfined Démarre dans le thread appelant, reprend n'importe où

Extensions

Fonctions d'extension

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

Propriétés d'extension

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

Surcharge d'opérateurs

```
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) // copie non destructive
val (name, age) = u1 // déstructuration
```

Membres auto-générés

equals() Égalité structurelle basée sur les propriétés
hashCode() Cohérent avec equals()
toString() `User(name=Alice, age=30)`
copy() Créer une copie modifiée
componentN() Support de la déstructuration

Classes enum

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

Classes sealed

Hiérarchie de classe sealed

```
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 exhaustif

```
fun handle(result: Result<String>): String = when (result) {
    is Result.Success -> result.data
    is Result.Error -> "Error: ${result.message}"
    is Result.Loading -> "Loading..."
} // pas de else nécessaire - le compilateur vérifie l'exhaustivité
```

Sealed vs Enum

Sealed class Les sous-classes peuvent avoir des états différents
Sealed interface Permet l'héritage multiple
Enum class Ensemble fixe d'instances singleton
data object Singleton avec surcharge de `toString()`

Fonctions de portée

Comparaison des fonctions de portée

let Contexte comme `it`, retourne le résultat du lambda
run Contexte comme `this`, retourne le résultat du lambda

with(obj) Contexte comme `this`, retourne le résultat du lambda
apply Contexte comme `this`, retourne l'objet contexte
also Contexte comme `it`, retourne l'objet contexte

let et apply

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

run et 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 est utile pour les effets de bord (journalisation, validation)
```