MANUAL

KOTLIN

IKER MARTÍN CAMBA

ÍNDICE

0.- Contexto

Este manual está desarrollado con el único objetivo de recopilar las líneas de código para el lenguaje de programación Kotlin.

Esta documentación no está pensada para el aprendizaje autónomo sin bases previas a dicho lenguaje.

No es un documento oficial, por lo que es posible que contenga algún error o inexactitud a la hora de especificar algún comando o línea de código.

1.- Variables

1.1.- Tipos de variables

```
val variable1 = "No editable"
const val constante = "No editable"

var variable2 = "Editable"


lateinit var variable3

variable3 = “Inicializar más tarde”
```

1.2.- Tipos de datos

```
var texto: String = "Contenido"
var booleano: Boolean = true
var decimal: Float = 9.99f
var entero: Int = 10
var doble: Double = 9.99 // Mayor capacidad que el Float
```

2.- Mostrar contenido en el terminal

```
println("Mostrar texto en la consola")

val ejemplo1: Int = 20
println("Contenido de variable: " + ejemplo1.toString())
```

3.- Operadores

3.1.- Matemáticos

```
var suma: Int = 2 + 4
var resta: Int = 10 - 5
var multiplicacion: Int = 2 * 6
var division: Int = 10 / 2
var resto: Int = 10 % 5

suma = suma + 1
suma++

resta = resta - 1
resta--
```

3.2.- Lógicos

```
valor1 == valor2   // Igual
valor1 != valor2   // Diferente
valor1 > valor2    // Mayor que
valor1 < valor2    // Menor que
valor1 >= valor2   // Mayor o igual
valor1 <= valor2   // Menor o igual
```

3.3.- Comparativos

```
condición1 && condición2    // Ambas deben ser TRUE
condición1 || condición2    // Al menos una debe ser TRUE
¡condición1                 // Debe ser lo contrario
```

4.- Condiciones

4.1.- IF

```
if (valor1 > valor2) {
    println("Valor 1 es mayor que valor 2")
}

if (valor1 > valor2) {
    println("Valor 1 es mayor que valor 2")
} else {
    println("Valor 1 es menor que valor 2")
}

if (valor1 > valor2) {
    println("Valor 1 es mayor que valor 2")
} else if (valor1 < valor2) {
    println("Valor 1 es menor que valor 2")
} else {
    println("Los valores son iguales")
}
```

4.2.- WHEN

```
val mes: Int = 6


when(mes) {
    1 -> println("Enero")
    2 -> println("Febrero")
    3,4,5,6,7,8,9,10,11,12 -> println("De Marzo a Diciembre")
    else -> println("Otro valor")
}
```

4.3.- DO WHILE

```
do {
    println("Se ejecuta la primera vez")
} while(valor1 == valor2)
```

4.4.- WHILE

```
while(valor1 == valor2) {
    println("Haz esto...")
}
```

4.5.- FOR

```
var nombres: Array<String> = arrayOf("Maria", "Ana", "Mikel")

for (i in nombres) {
    println("Nombre: " + i)
}
```

5.- Funciones

5.1.- Básica

```
fun mostrar_saldo() {
    println("Tu saldo es desconocido...")
}


mostrar_saldo()    // Ejecutar la función
```

5.2.- Con parámetros

```
fun sumar_valores(primer_valor: Int, segundo_valor: Int) {
    var resultado: Int = primer_valor + segundo_valor
    println("La suma de los valores es: " + resultado)
}


sumar_valores(2,5)     // Ejecutar la función
```

5.3.- Con retorno

```
fun obtener_dinero(tengo_dinero: Boolean): Boolean {
    if (!tengo_dinero) {
        return false
    } else {
        return true
    }
}


var tengo_algo: Boolean = obtener_dinero(false)
```

6.- Estructuras de datos

6.1.- LIST

```
val lista: List<String> = listOf("A", "B", "C")

var obtenerPrimerValor = lista[0]
```

6.2.- MUTABLE LIST

```
val listaMutable: MutableList<String> = mutableListOf("A",
"B", "C")

listaMutable.add("D")            // Añadir contenido
listaMutable.remove("B")         // Eliminar contenido

var obtenerPrimerValor = listaMutable[0]
```

6.3.- ARRAY

```
val array: Array<String> = arrayOf("A", "B", "C")

array[1] = "D"                   // Añadir contenido

var obtenerPrimerValor = array[0]
```

6.4.- SET

```
val set: Set<String> = setOf("A", "B", "C")

val contieneElValor = set.contains("A")
```

6.5.- MUTABLE SET

```
val setMutable: MutableSet<String> = mutableSetOf("A", "B",
"C")

setMutable.add("D")                    // Añadir contenido
setMutable.remove("B")                 // Eliminar contenido

val contieneElValor = setMutable.contains("A")
```

6.6.- MAP

```
val mapa: Map<String, Int> = mapOf("A" to 1, "B" to
2, "C" to 3)

val valorDelA = mapa["A"]
```

6.7.- MUTABLE MAP

```
val mapaMutable: MutableMap<String, Int> = mutableMapOf("A" to 1, "B" to 2, "C" to 3)

mapaMutable["D"] = 4              // Añadir contenido
mapaMutable.remove("B")           // Eliminar contenido

val ValorDelC = mapaMutable["C"]
```

7.- Objetos

7.1.- Atributos y métodos

```
class Persona(){

    var alive: Boolean = true        // Atributo

    fun die() {                      // Función o método
        this.alive = false
    }
}
```

```
var persona1: Persona = Persona()
println(persona1.alive)          // TRUE
persona1.die()
println(persona1.alive)          // FALSE
```

7.2.- Constructores

Los constructores permiten especificar atributos a la hora de la creación del objeto.

```
class PersonaV2(var nombre: String, var edad: Int){  }

class PersonaV3(var nombre: String = "Marcos",
                var edad: Int = 18){ }
```

```
var persona2: PersonaV2 = PersonaV2("Antonio", 22)
println(persona2.nombre)          // Antonio

var persona3: PersonaV3 = PersonaV3()
println(persona3.nombre)          // Marcos
```

7.3.- Encapsulamiento

Establecer los atributos de un objeto como privados y crear métodos para obtener y modificar dichos valores.

```
class PersonaV4(private var nombre: String,
                private var edad: Int){
    // Función para editar valores
    fun porDefecto() {
        this.nombre = “Nombre X”
        this.edad = 18
    }
    // GETTER -> Obtener valores
    fun getNombre(): String {
        return this.nombre
    }
    // SETTER -> Cambiar valores
    fun setNombre(nuevoNombre: String) {
        this.nombre = nuevoNombre
    }
}
```

```
var persona4: PersonaV4 = PersonaV4(“anTonio”, 20)
println(persona4.getNombre())   // anTonio
persona4.setNombre(“Antonio”)
println(persona4.getNombre())   // Antonio
```

7.4.- Herencia y override

```
class Persona(val nombre: String, val edad: Int) {
    fun mostrarInformacion() {
        println("Primera fun")
    }
}

// Clase que hereda de Persona
class Estudiante(nombre: String, edad: Int, val escuela:
String ) : Persona(nombre, edad) {
    // Sobrescribe el método mostrarInformacion original
    override fun mostrarInformacion() {
        super.mostrarInformacion()
        println("Segunda fun")
    }
}
```

```
val persona = Persona("Juan", 40)
persona.mostrarInformacion()    // Primera fun

val estudiante = Estudiante("Ana", 20, “UPV”)
estudiante.mostrarInformacion() // Primera fun + Segunda fun
```

7.5.- Public, Protected, Private e Internal

TIPO	ACCESO DISPONIBLE EN
public	En cualquier lugar
private	Solo en la clase donde se define
protected	En la clase y sus subclases (herencia)
internal	Dentro del mismo módulo, que es una parte de un proyecto (app, library, network, etc)

7.6.- Interfaces

Es una estructura que define un conjunto de funciones y propiedades que una clase debe implementar y no hace falta definir cómo funcionan exactamente. Permite que una clase pueda heredar de varias interfaces.

```
interface Solido {
    fun soySolido() {
        println("Soy solido")
    }
}

interface Liquido {
    var data: Int
    fun soyLiquido() {
        println("Soy liquido")
    }
}

class Estado(): Solido, Liquido {
    override var data: Int = 0
}
```

8.- Tipos de clases

TIPO	DESCRIPCIÓN
abstract	Define métodos o propiedades sin implementación (requiere subclases)
anidada	Clase contenida dentro de otra, pero no tiene acceso a los miembros de la clase externa.
inner (interna)	Clase que puede acceder a los miembros de la clase externa.
enum	Representa un conjunto de constantes predefinidas
data	Almacena datos con métodos generados automáticamente

8.1.- Abstracta

```
abstract class Vehiculo {
    abstract fun drive() // Método sin implementación

    fun stop() {
        println("El vehículo se ha detenido")
    }
}

class Coche : Vehiculo() {
    override fun drive() {
        println("El coche está en marcha")
    }
}
```

```
val miCoche = Coche()
miCoche.drive()         // El coche está en marcha
miCoche.stop()          // El vehículo se ha detenido
```

8.2.- Anidada

```
class Contenedor {
    private var mensaje = "Soy el contenedor"
    fun presentar(): String {
        return this.mensaje
    }

    class Anidada {
        fun presentar(): String {
            return mensaje
        }
    }
}
```

```
var contenedor = Contenedor()
println(contenedor.presentar())      // Soy el contenedor

var anidada = Contenedor.Anidada()
println(anidada.presentar())         // Soy el contenedor
```

8.3.- Interna

```
class Contenedor {
    private var mensaje = "Soy el contenedor"
    fun presentar(): String {
        return this.mensaje
    }

    inner class Interna {
        private var mensaje = "Soy la interna"
        fun presentar(): String {
            return mensaje
        }
    }
}
```

```
var contenedor = Contenedor()
println(contenedor.presentar())      // Soy el contenedor

var interna = Contenedor().Interna()
println(interna.presentar())         // Soy la interna
```

8.4.- Data

```
data class Persona(val nombre: String, val dni: String) {  }
```

```
val persona1 = Persona("Juan",12345678A)
val persona2 = Persona("Juan",12345678A)
println(persona1)      // Persona(nombre=Juan, dni=12345678A)

val persona3 = persona1.copy(nombre = "Ana")
println(persona3)      // Persona(nombre=Ana, dni=12345678A)
```

8.5.- Enum

```
enum class Dias {
    LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO;

    fun saludo(): String {
       when (this) {
           LUNES -> return "Es Lunes"
           MARTES -> return "Es Martes"
           MIERCOLES -> return "Es Miercoles"
           JUEVES -> return "Es Jueves"
           VIERNES -> return "Es Viernes"
           SABADO -> return "Es Sabado"
           DOMINGO -> return "Es Domingo"
       }
    }
}
```

```
var hoy: Dias = Dias.SABADO
var semana = Dias.values()
for (i in semana) {
    println(i)
}

println(hoy.name)           // SABADO
println(hoy.ordinal)        //5
println(hoy.saludo())       // Es sabado
```

9.- Tipos de funciones

9.1.- Extensión

```
private fun String.noSpaces(): String{
    return this.replace(" ", "")
}


var conEspacios = "1 23 456 7 8 9"
println(conEspacios.noSpaces())      // 123456789
```

9.2.- Orden superior

```
private fun calculadora(n1: Int, n2: Int,
                        fn: (Int, Int)->Int): Int {
    return fn(n1, n2)
}

private fun suma(x: Int, y: Int): Int {
    return x + y
}

private fun resta(x: Int, y: Int): Int {
    return x - y
}
```

```
println("La suma de 8 y 2 es: " + calculadora(8, 2, ::suma))
```

9.3.- Lambda

```
private fun calculadora(n1: Int, n2: Int,
                        fn: (Int, Int)->Int): Int {
    return fn(n1, n2)
}


var funcion = { x: Int, y: Int -> x + y }

println("La suma de 8 y 2 es: " + calculadora(8, 2, funcion))
```

```
println("La potencia de 2 elevado a 5 con lambda es: " +
calculadora(2, 5,
    { x, y ->
        var valor = 1
        for (i in 1..y) valor += x
        valor      // Al final se tiene que indicar el valor
    }
))
```

9.4.- Scope

Se encargan de llevar a cabo la ejecución de un bloque de código sobre un objeto.

TIPO	REFERENCIA	OBJETIVO
let	it	Trabajar con objetos no nulos y transformar.
run	this	Ejecutar operaciones complejas.
with	this	Realizar operaciones en el mismo objeto.
apply	this	Realizar operaciones en el mismo objeto.
also	it	Operaciones adicionales sin modificar.

9.4.1.- LET

```
val nombre = "Kotlin"
val longitud = nombre.let {
    println("El nombre es: $it")    // El nombre es: Kotlin
    it.length
}

println("Longitud: $longitud")      // Longitud: 6
```

9.4.2.- RUN

```
val resultado = "Kotlin".run {
     println("Trabajando con $this")
     length        // Devuelve la longitud del String
}

println("Longitud: $resultado")      // Longitud: 6
```

9.4.3.- WITH

```
val persona = "Ana"
val mensaje = with(persona) {
     println("Hola, $this")
     length
}

println("Longitud del nombre: $mensaje")
// Longitud del nombre: 3
```

9.4.4.- APPLY

```
data class Persona(var nombre: String, var edad: Int)
val persona = Persona("Ana", 25).apply {
     nombre = "Ana María"
     edad = 26
}

println(persona)       // Persona(nombre=Ana María, edad=26)
```

9.4.5.- ALSO

```
val nombre = "Kotlin".also {
    println("El nombre es $it")
}

println("Nombre final: $nombre")     // Nombre final: Kotlin
```

10.- Typealias

```
typealias aliasDato = MutableMap<Int, ArrayList<String>>


var mutableMap2: aliasDato = mutableMapOf()
mutableMap2[0] = arrayListOf("Hola", "Adios")
```

11.- Desestructuración

```
data class Estrella(val nombre: String, val radio: Float,
                    val galaxia: String) {   }
```

```
var (n, r, g) = Estrella("Sol", 69f, "Via láctea")


var (n2, _, g2) = Estrella("Sol", 69f, "Via láctea")


var componente = Estrella("Sol", 69f, "Via láctea")
println("Valor 1: " + componente.component1())
println("Valor 2: " + componente.component2())
println("Valor 3: " + componente.component3())
```

12.- Try catch

```
try {
    println("Posible error....")
} catch (e: Exception) {
    println("Error: " + e)
} finally {
    println("Esto se va a ejecutar si o si, siempre")
}
```