UTT
UNIVERSIDAD TECNOLÓGICA DE TIJUANA
GOBIERNO DE BAJA CALIFORNIA

# Investigación Kotlin


**PRESENTADO POR:**

García Arreola Howard Isaí

**GRUPO:** 9B


**MATERIA:**

Desarrollo para Dispositivos Inteligentes

**PROFESOR:**

Ray Brunett Parra Galaviz

**FECHA:**

02 de octubre de 2024

## 1. Sentencias de Iteración

Las sentencias de iteración, o bucles, permiten repetir un bloque de código mientras se cumple una determinada condición o se recorren elementos de una colección.

### 1.1. for

**Ejemplo:**

```
for (i in 1..5) {
    println(i)
}
```

Este bucle imprime los números del 1 al 5. El operador **“..”** crea un rango inclusivo. También puedes usar **until** para hacer un rango exclusivo del límite superior:

```
for (i in 1 until 5) {
    println(i)  // Imprime del 1 al 4
}
```

**Iterando sobre una lista**

```
val lista = listOf("Kotlin", "Java", "Python")
for (elemento in lista) {
    println(elemento)
}
```

### 1.2. while y do-while

El bucle while repite un bloque de código mientras una condición sea verdadera, mientras que el do-while asegura que el bloque se ejecute al menos una vez antes de verificar la condición.

**Ejemplo:**

```
var i = 1
while (i <= 5) {
    println(i)
    i++
```

```
}
```

Este ejemplo imprime los números del 1 al 5 mientras i sea menor o igual a 5.

**Ejemplo: do-while**

```
var j = 6
do {
    println(j)
    j++
} while (j <= 5)
```

Se ejecuta una vez aunque la condición no se verdadera.

**2. Sentencias de Control**

Las sentencias de control son estructuras que permiten modificar el flujo de ejecución de un programa en función de condiciones lógicas.

**2.1. Sentencia if**

```
val x = 10
if (x > 5) {
    println("x es mayor que 5")
} else {
    println("x no es mayor que 5")
}
```

**2.2. Sentencia when**

La sentencia when es una versión del switch en otros lenguajes y puede utilizarse tanto como una expresión o declaración.

**Ejemplo básico de when:**

```
val dia = 3
when (dia) {
    1 -> println("Lunes")
    2 -> println("Martes")
    3 -> println("Miércoles")
    else -> println("Día desconocido")
}
```

También puede evaluar condiciones complejas:

```
val numero = 5
when {
    numero % 2 == 0 -> println("Es un número par")
    else -> println("Es un número impar")
}
```

**2.3. Sentencias break y continue**

- **break:** Termina el bucle inmediatamente.
- **continue:** Salta la iteración actual y pasa a la siguiente.

**Ejemplo:**

```
for (i in 1..10) {
    if (i == 5) break  // Termina el bucle cuando i es 5
    println(i)
}

for (i in 1..10) {
    if (i % 2 == 0) continue  // Salta los números pares
    println(i)
}
```

**3. Funciones en Kotlin**

Las funciones en Kotlin encapsulan bloques de código que pueden ser reutilizados. Pueden tener parámetros, devolver valores y son capaces de realizar una gran variedad de operaciones.

**3.1. Declaración de funciones**

Para declarar una función, se usa la palabra clave **fun**.

**Ejemplo:**

```
fun saludar(nombre: String) {
    println("Hola, $nombre")
}
```

**3.2. Funciones con valor de retorno**

Las funciones pueden devolver valores. El tipo de retorno se define después de los parámetros.

**Ejemplo:**

```
fun sumar(a: Int, b: Int): Int {
    return a + b
}

fun main() {
    val resultado = sumar(3, 4)
    println(resultado)  // Imprime 7
}
```

### 3.3. Funciones de una sola expresión

Kotlin permite definir funciones de una sola expresión de manera concisa sin usar return.

```
fun multiplicar(a: Int, b: Int) = a * b
```

### 3.4. Parámetros por defecto y parámetros nombrados

Kotlin soporta parámetros con valores por defecto y también llamados por su nombre.

**Ejemplo:**

```
fun saludo(nombre: String = "Desconocido", edad: Int = 0) {
    println("Hola, $nombre. Tienes $edad años.")
}

fun main() {
    saludo("Carlos", 25)  // Llama con parámetros
    saludo()  // Usa los valores por defecto
}
```

### 3.5. Funciones de orden superior y lambdas

Kotlin soporta funciones de orden superior, lo que significa que puedes pasar funciones como parámetros o devolverlas.

**Ejemplo de función de orden superior:**

```
fun operar(a: Int, b: Int, operacion: (Int, Int) -> Int): Int {
    return operacion(a, b)
```

```
}

fun main() {
    val suma = operar(5, 3, { x, y -> x + y })
    println(suma)  // Imprime 8
}
```