Lenguaje: Kotlin

1. Introducción
2. Aspectos básicos
3. Bucles
4. Clases

Introducción

HOLA KOTLIN

Kotlin es un lenguaje cómodo para cualquiera que sepa Java

```
package holaKotlin

fun main(args: Array) {
    println("Hola Kotlin!")
}
```

Usar Kotlin para desarrollar Android

- Compatibilidad
- Simplicidad
- Menos errores
- Alto nivel
- Sintaxis Limpia

Documentación Kotlin

Aspectos básicos

2. Aspectos básicos

Variables

Strings

Condicionales

Funciones

Valores Null

2. Aspectos básicos

Variables

Condicionales

Strings

Funciones

Valores Null

VARIABLES – **val / var**

```
// Variables de lectura - identificador val
val a: Int = 1

val b: Int // Si no se inicializa se debe definir el tipo
b = 2

// Variables que se pueden modificar - identificador var
var c = 3
c = c + 4
```

2. Aspectos básicos

Variables

Funciones

Strings

Condicionales

Valores Null

FUNCIONES - **fun**

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

// o

fun suma(a: Int, b: Int) = a + b
```

FUNCIONES - **fun**

```
fun pedirUnaPizza(queso: Boolean = true, jamon: Boolean = false,
                  aceitunas: Boolean = false, tomate: Boolean = true,
                  cebolla: Boolean = false, extraQueso: Boolean = false)
{
...
}

// Para pedir una pizza jamón y queso simplemente llamamos la función:

pedirUnaPizza(jamon = true)

// Y la pizza va a llevar queso, tomate y jamón
```

2. Aspectos básicos

Variables

Strings

Condicionales

Funciones

Valores Null

STRINGS

```
var a = 1

val s1 = "El valor de a es: $a"

println(s1)
println(s1.subSequence(3,8))
println(s1.contains("valor", ignoreCase = false)
//También se puede escribir como:
println("valor" in s1)

/* Resultado:
El valor de a es: 1
valor
true
*/
```

Variables

Condicionales

Strings

Funciones

Valores Null

CONDICIONALES

```
fun max(a: Int, b: Int): Int {
     if (a > b) {
          return a
     } else {
          return b
     }
}

// o

fun max(a: Int, b: Int) = if (a > b) a else b
```

CONDICIONALES

when: reemplaza el "switch" de los lenguajes basados en C.

```
val foo = listOf(7,8,9)

when (a) {
     1 -> print("a == 1!")
     2 -> print("a == 2!")
     in 3..5 -> print("a está entre 3 y 6")
     in foo -> print("a está entre 7 y 9")
     else -> {
          print("ninguna de las anteriores")
     }
}
```

Variables

Condicionales

Strings

Funciones

Valores Null

VALORES NULL

- Todas las declaraciones son non-null. Para declarar una variable que pueda contener valores null se debe indicar con el identificador **?**

```
// Función que devuelve null si la variable str no se puede convertir a número

fun parseInt(str: String): Int? {
     return str.toIntOrNull()
}
```

VALORES NULL

```
var noNull: String "Hola"
var puedeSerNull: String? = null // "?" indica que puede ser null

if (noNull.length > 0) { // No va a fallar por null
     ...
}

if (puedeSerNull.length > 0) { // El compilador controla por ti el error si es null
    ...
}
```

Bucles

ITERACIONES BÁSICAS

For

While

Do while

ITERACIONES BÁSICAS

For

While

Do while

FOR

for : bucle para recorrer un elemento del tipo iterable. Ej: Lista

Listas en Kotlin

```
val coches = listOf("Mazda", "Jeep", "Seat")
for (coche in coches) {
     println(coche)
}

/* Resultado:
Mazda
Jeep
Seat
*/
```

ITERACIONES BÁSICAS

For

While

Do while

WHILE

while: bucle para recorrer hasta cumplir la expresión contenida.

```
val coches = listOf("Mazda", "Jeep", "Seat")
var indice = 0
while (indice < coches.size) {
      println("El coche en la posición $indice es: ${coches[indice]}")
      indice++
}

/* Resultado:
El coche en la posición 0 es: Mazda}
El coche en la posición 1 es: Jeep}
El coche en la posición 2 es: Seat}
*/
```

ITERACIONES BÁSICAS

For

While

Do while

DO WHILE

do while: bucle para recorrer hasta cumplir la expresión contenida, pero siempre se ejecuta mínimo una vez.

Operadores Kotlin

```
var numero: Int

do {
     println("Introduce un número entre 1 y 5")
     numero = readLine()!!.toInt()
} while(numero !in 1..5)
```

Clases

- Se declaran con la palabra **class**.
- Tienen un constructor primario y uno o más constructores secundarios.
    - El constructor primario puede formar parte del encabezado de la clase.
- Se pueden declarar los atributos de la clase en el propio constructor.

```
class Persona(nombre: String, edad: Int)
{

...

}
```

```java
public class Persona {
    private String nombre;
    private int edad;

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getEdad() {
        return edad;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }
}
```

- Exemple de classe en Java

- Si especificamos los atributos de la clase en la declaración de la misma, deberemos definirlos dentro de un bloque **init**.

```
class Persona(nombre: String, edad: Int)
{
     init {
          println("Inicialización de Persona");
     }
}
```

- Para una sitaxis más entendible, se recomienda hacer uso de **constructores específicos** o **secundarios**:

```
class Persona {
	private var nombre: String
	private var edad: Int

	constructor(nombre: String, edad: Int) {
		this.nombre = nombre
		this.edad = edad
	}
}
```

- Para crear una nueva instancia o objeto, se llama a uno de los constructores.

```
val pitter = Persona("Pitter", 15)
```

- No se utiliza la palabra clave **new** para reservar memória para el nuevo objeto.

"Medir el progreso de la programación por líneas de código es como medir el progreso en la construcción de aviones por el peso."

BILL GATES, Co-fundador de Microsoft.