Seminario de Lenguajes opción Android + Kotlin
Tutorial básico de Kotlin

Introducción

Kotlin es un lenguaje de programación moderno y expresivo que se ha convertido en el lenguaje de programación recomendado para Android. Es un lenguaje de programación orientado a objetos y funcional que presenta una sintaxis clara y concisa que lo hace fácil de leer y escribir.

Tipos de datos

En Kotlin, hay dos tipos de datos principales: tipos de **datos primitivos** y tipos de **datos de objetos**.

1. Los tipos de datos primitivos incluyen:
    - **Boolean**: representa un valor booleano true o false.
    - **Char**: representa un carácter Unicode de 16 bits.
    - **Byte**: representa un número entero de 8 bits con signo.
    - **Short**: representa un número entero de 16 bits con signo.
    - **Int**: representa un número entero de 32 bits con signo.
    - **Long**: representa un número entero de 64 bits con signo.
    - **Float**: representa un número en coma flotante de 32 bits.
    - **Double**: representa un número en coma flotante de 64 bits.

2. Los tipos de datos de objetos incluyen:
    - **String**: representa una cadena de caracteres.
    - **Array**: representa un arreglo de elementos.
    - **Collection**: representa una colección de elementos.
    - **List**: representa una lista ordenada de elementos.
    - **Set**: representa un conjunto de elementos únicos.
    - **Map**: representa un mapa de pares clave-valor.

Además de estos tipos de datos predefinidos, Kotlin también admite la creación de tipos de datos personalizados, utilizando las clases y estructuras de datos personalizadas.

Variables

Para declarar una variable en Kotlin, se utiliza la palabra clave **var** seguida del nombre de la variable y su tipo, separados por dos puntos. Por ejemplo, para declarar una variable entera llamada “edad”, se puede escribir:

```
var edad: Int = 25
```

En este ejemplo, “edad” se declara como una variable entera con un valor inicial de 25.

Kotlin también admite la **inferencia de tipos de datos**. Esto significa que si no se especifica el tipo de la variable, Kotlin puede deducir del valor inicial que se le asigne. Por ejemplo, se puede declarar una variable entera como:

```
var edad = 25
```

En este ejemplo, Kotlin infiere que “edad” es una variable entera porque se le ha asignado un valor entero.

Constantes

En Kotlin, se puede declarar una constante utilizando la palabra clave **val** en lugar de var. Las constantes son variables que no se pueden cambiar después de su inicialización. Por ejemplo, se puede declarar una constante como:

```
val pi = 3.14159
```

En este ejemplo, pi es una constante que se inicializa con el valor de Pi.

Null Safety

Una de las características más importantes de Kotlin es la null safety, que ayuda a **prevenir errores** comunes **relacionados con valores nulos**. En Kotlin, una variable no puede tener un valor nulo a menos que se le indique explícitamente que puede serlo. Por ejemplo, se puede declarar una variable que pueda ser nula como:

```
var mensaje: String? = null
```

En este ejemplo, “mensaje” es una variable que puede ser nula porque se le ha asignado el valor nulo.

Para acceder a una propiedad o método de una variable que puede ser nula, se debe utilizar el operador de llamada segura “**?**”. Por ejemplo, si se quiere llamar al método *length* en la variable “mensaje”, se debe escribir:

```
val longitud = mensaje?.length
```

En este ejemplo, si “mensaje” es nulo, la expresión entera devuelve nulo. Si “mensaje” no es nulo, la expresión devuelve la longitud de la cadena.

Funciones

Las funciones son bloques de código que realizan una tarea específica. En Kotlin, se puede declarar una función utilizando la palabra clave **fun**. Por ejemplo, se puede declarar una función que calcule la suma de dos números enteros como:

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

En este ejemplo, “suma” es una función que toma dos argumentos enteros, a y b, y devuelve su suma.

Estructuras de control

En Kotlin, existen varias **estructuras de control** que se utilizan para **tomar decisiones** y **repetir bloques de código** según ciertas condiciones. Algunas de las estructuras de control más comunes son:

1. **if-else**: se utiliza para ejecutar diferentes bloques de código dependiendo de si una condición es verdadera o falsa. Ejemplo:

```
val numero = 10;
if (numero > 0) {
   println("El número es positivo")
} else {
   println("El número es negativo o cero")
}
```

2. **when**: se utiliza para evaluar una expresión y ejecutar diferentes bloques de código según el valor de la expresión. Es similar a un *switch* en otros lenguajes de programación. Ejemplo:

```
val dia = 3;
when (dia) {
   1 -> println("Lunes")
   2 -> println("Martes")
   3 -> println("Miércoles")
   4 -> println("Jueves")
   5 -> println("Viernes")
   else -> println("Fin de semana")
}
```

3. **for**: se utiliza para repetir un bloque de código varias veces, generalmente para iterar sobre una colección o un rango de valores. Ejemplo:

```
val numeros = listOf(1, 2, 3, 4, 5);
for (numero in numeros) {println(numero)}
```

4. **while y do-while**: se utilizan para repetir un bloque de código mientras se cumpla una condición. La diferencia entre ambos es que **while verifica la condición antes** de ejecutar el bloque de código, mientras que **do-while verifica la condición después** de ejecutar el bloque de código al menos una vez. Ejemplo:

```
var contador = 0;
while (contador < 10) {
```

```
   println(contador);
   contador++;
};
do {
   println(contador);
   contador--
} while (contador > 0)
```