Manual de Introducción a Kotlin

1. Introducción a Kotlin

Kotlin es un lenguaje de programación moderno y expresivo que se ejecuta en la máquina virtual de Java (JVM). Fue desarrollado por JetBrains y se utiliza ampliamente en el desarrollo de aplicaciones Android, aunque también es popular para el desarrollo de aplicaciones del lado del servidor, aplicaciones multiplataforma, y más.

2. Variables

En Kotlin, las variables pueden declararse de dos maneras: usando `val` o `var`.

- `val`: define una variable de solo lectura (inmutable). Una vez asignado un valor, no se puede cambiar.
- `var`: define una variable mutable, cuyo valor puede cambiarse después de su asignación inicial.

```
val nombre: String = "Alberto"  // Inmutable
var edad: Int = 35              // Mutable
```

- **Inferencia de tipos**: Kotlin puede inferir el tipo de una variable a partir de su valor, por lo que la declaración de tipos puede ser opcional.

```
val apellido = "Rodríguez" // Kotlin infiere que es un String
var altura = 1.75      // Kotlin infiere que es un Double
```

- No siempre kotlin infiere el tipo de variable, para declarar el tipo de variable que es:

```
val edad:Int = 40
```

- Las variables numéricas pueden ser de diferentes tipos:

```
//Int -2,147,483,647 a 2,147,483,647

//Long

//Float - Terminan con f

//Double - Admite mas decimales que float. Solo en caso de necesitar algo super potente com
```

- Las variables alfanuméricas son variables que contienen valoras alfanuméricos además de números

```
//Variables alfanuméricas
```

```
//Char: Solo soporta un caracter
val charEjemplo1:Char = 'e'
val charEjemplo1:Char = '2'
val charEjemplo1:Char = '@'

//String: Soporta lo que sea y de tamaño que queramos. Va con comillas dobles
val stringEjemplo:String = "Bienvenidos a DAM 2"
```

- Variables Booleanas

```
//Variables booleanas
val boolEjemplo1:Bool = true
val boolEjemplo2:Bool = false
```

- Los val no pueden ser reasignados, no se les puede cambiar el valor. `val` es una constante. Si queremos cambiar el valor de las variables hemos de usar `var`
- La diferencia entre la orden print o println es que ala hora de imprimir un resultado noslo imprime en una linea nueva

3. Funciones

Las funciones en Kotlin se definen usando la palabra clave `fun`. A continuación, se muestra la estructura básica de una función:

```
fun nombreFuncion(parametro1: Tipo, parametro2: Tipo): TipoDeRetorno {
    // Cuerpo de la función
    return valorDeRetorno
}
```

Ejemplo de una función que suma dos números enteros:

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

- **Funciones de una sola expresión**: Si la función es simple, puedes definirla en una sola línea.

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

4. Instrucciones `if-else`

La estructura `if-else` en Kotlin es similar a la de otros lenguajes, pero en Kotlin también puede usarse como una expresión.

Uso básico

```
if (condicion) {
    // Código si la condición es verdadera
```

```
} else {
    // Código si la condición es falsa
}
```

`if` como expresión

En Kotlin, `if` puede retornar un valor, lo que permite asignar el resultado a una variable.

```
val max = if (a > b) a else b
```

5. Expresión `when`

La expresión `when` en Kotlin es una alternativa más potente y flexible al `switch` de otros lenguajes. Permite evaluar una variable o expresión y ejecutar el bloque de código correspondiente al valor coincidente.

Uso básico

```
when (variable) {
    valor1 -> {
        // Código para valor1
    }
    valor2 -> {
        // Código para valor2
    }
    else -> {
        // Código si no coincide ningún valor
    }
}
```

Uso como expresión

`when` también puede retornar un valor, de forma similar a `if`.

```
val resultado = when (variable) {
    1 -> "Uno"
    2 -> "Dos"
    else -> "Otro número"
}
```

6. Arrays y Listas

Arrays

Un array en Kotlin se define utilizando la función `arrayOf` o con un tipo específico.

```kotlin
val numeros = arrayOf(1, 2, 3, 4, 5)
val nombres = arrayOf("Juan", "María", "Luis")
```

Listas

En Kotlin, existen dos tipos principales de listas:

- **List**: es inmutable, es decir, no se puede modificar después de su creación.
- **MutableList**: se puede modificar después de su creación.

```kotlin
val listaInmutable = listOf(1, 2, 3)  // No se puede modificar
val listaMutable = mutableListOf(1, 2, 3)  // Se puede modificar
```

Operaciones comunes en listas

Agregar un elemento a una lista mutable:

```kotlin
listaMutable.add(4)
```

Acceder a un elemento de una lista:

```kotlin
val primerElemento = listaInmutable[0]
```
