Kotlin

Cesar Camilo Izquierdo Gallardo - 1152251
Omar David Jaimes Molina - 1152263
Omar Alexis Palencia Claro - 1152270

Introducción

Kotlin data de 2016, cuando JetBrains, la famosa empresa checa creadora de varios de los IDE's más usados como IntelliJ, WebStorm entre otros, publica la primera versión de este lenguaje de programación.

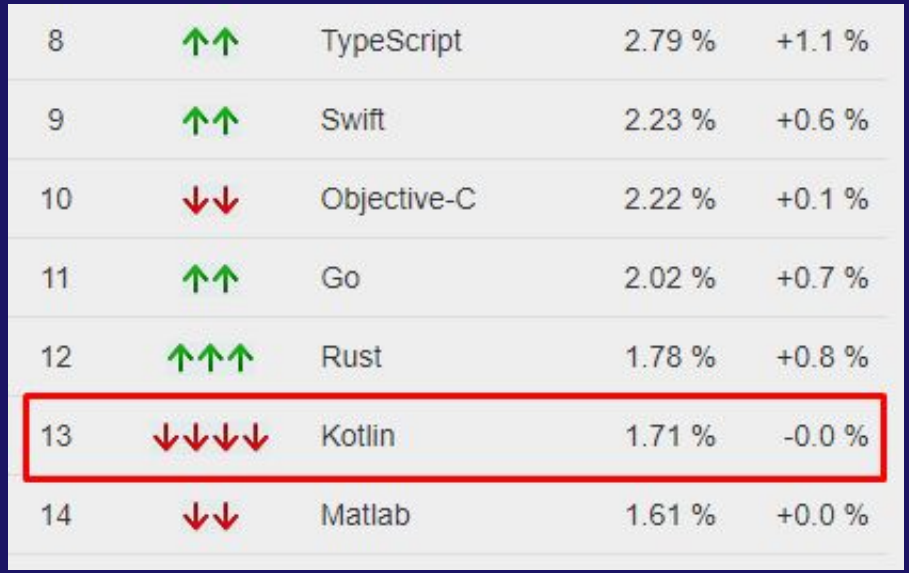
Version actual: Kotlin 1.7.21

“Kotlin First”

–Google

Ubicación en el ranking de lenguajes:

El Índice de popularidad de lenguajes de programación PYPL se crea analizando la frecuencia con la que se buscan tutoriales de idiomas en Google.



8	↑↑	TypeScript	2.79 %	+1.1 %
9	↑↑	Swift	2.23 %	+0.6 %
10	↓↓	Objective-C	2.22 %	+0.1 %
11	↑↑	Go	2.02 %	+0.7 %
12	↑↑↑	Rust	1.78 %	+0.8 %
13	↓↓↓↓	Kotlin	1.71 %	-0.0 %
14	↓↓	Matlab	1.61 %	+0.0 %

Sintaxis de Kotlin:

```
//Importar clases y paquetes

package com.example

import com.example

//Declaración de variables

//Variable de tipo final
val saludar = “¡Hola mundo!”//Detecta que
es un string
val saludar: String = “!Hola mundo¡”

//Variable que puede cambiar
var año = 2022//Detecta que es un entero
var año: Int = 2022
```

Sintaxis de Kotlin:

```
//Funciones

//Función sin retorno
fun mostrarSuma (a: Int, b: Int){
    println("La suma de $a y $b es:
            ${a + b}")
}

//Función con retorno
fun suma (a: Int, b: Int): Int {
    return a + b
}

//Otra forma de escribir una función
fun suma (a: Int, b: Int) = a + b
```

Sintaxis de Kotlin:

```
//Estructuras condicionales

//if else
fun numeroMayor (a: Int, b: Int): Int {
    var mayor: Int = 0
    if (a > b) {
        mayor = a
    } else {
        mayor = b
    }
    return mayor
}

//Una forma de ahorrar código
fun numeroMayor (a: Int, b: Int) =
if (a > b) a else b
```

Sintaxis de Kotlin:

```
//Estructuras condicionales

//when
fun describir(algo: Any): String =
    when (algo) {

        1 → "Uno"

        "hola" → "Saludos"

        is Int → "Número entero"

        !is String → "No es un texto"

        else → "Desconocido"
    }
```

Sintaxis de Kotlin:

```
//Estructuras repetitivas

//for
for (i in 1..3) {
    print("$i ") // "1 2 3"
}

for (i in 10 downTo 0 step 2) {
    print("$i ") // "10 8 6 4 2 0"
}

//for para recorrer un arreglo
val frutas = listOf("platano", "manzana",
"pera", "kiwi")
for (index in frutas.indices) {
    println("La fruta en la posición $index
es ${frutas[index]}")
}
```

Sintaxis de Kotlin:

```
//Estructura repetitivas

//while

var numero = 0
while (numero != 2) {
    numero = Random.nextInt(0,6)
    print("$numero ")
}

//Arrays
val numerosConDecimales = arrayOf<Double>(
    0.5, 2.5, 4.0, 5.0,
    4.5, 6.0, 7.6, 8.0,
    5.0, 6.4, 4.0, 9.1
)
```

Sintaxis de Kotlin:

```
//Modificadores de acceso

public: este es el modificador de acceso por
defecto. Con este modificador el elemento
puede ser accedido desde cualquier parte
del proyecto.

private: con este modificador, la clase sólo
puede ser accedido por las clases miembro

protected: disponibles en la clase miembro
y sub-clases de la clase

internal: con este modificador, la clase
puede ser accedida por cualquiera dentro
del módulo.
```

Sintaxis de Kotlin:

```
//Clases y Objetos

//Clase vacia
class Empty

//Clase con constructor
class Persona {
    var nombres: String

    constructor (nombre: String) {
        this.nombre = nombre
    }
}

//Los getter y setter son generados
automáticamente por el compilador de
Kotlin, sin embargo, podemos reescribirlos.
```

Sintaxis de Kotlin:

```
//Clases y Objetos

//Toda clase en Kotlin tiene como
superclase Any, de la misma manera que en
Java con Object

//La clase Any tiene los métodos equals,
hashCode y toSting.

fun equals(other: Any?): Boolean

fun hashCode(): Int

fun toString(): String
```

Sintaxis de Kotlin:

```
//Clases y Objetos

//Para heredar una clase usamos : y la palabra reservada open para dar capacidad de herencia

open class Persona

clas Hijo : Persona()

//Para crear un nuevo objeto se hace de la siguiente forma:

val futbolista = Persona()
```

Sintaxis de Kotlin:

```
//Polimorfismo

//Para que la herencia múltiple usamos las
interfaces al igual que en java
interface Interfaz1{
    fun metodo1(){
        println("metodo 1")
    }
}
open class ClasePadre{
    fun metodo2(){
        println("metodo 3")
    }
}
class ClaseHeredera: ClasePadre(), Interfaz1,
Interfaz2{
    fun metodo3(){
        println("metodo 4")
    }
}
```