

# Kotlin

<http://bit.ly/KotlinCalculoGitHub>

<https://play.kotlinlang.org/>

---

Daniel Illescas Romero

15 de noviembre de 2018

Calculo S.A

## 1. Introducción

## 2. Sintaxis y ejemplos

Variables y constantes

Funciones

Clases

Polimorfismo

For, Map, Range, List, When

## 3. Interoperabilidad

## 4. Avanzado

## 5. Bibliografía

# Introducción

---

# Qué es

- **Kotlin** es un lenguaje de propósito general, estáticamente tipado, que combina orientación a objetos con programación funcional.
- Funciona donde **Java** funcione: apps móviles, aplicaciones de escritorio, apps del lado del servidor.
- Tiene **soporte** oficial para:
  - **IntelliJ IDEA** y Android Studio
  - Eclipse
  - Maven, Gradle, Ant
  - Spring Boot
  - etc [1]

## Quién está detrás

- Es un lenguaje **open-source**, disponible en GitHub y con más de 250 contribuyentes.
- **JetBrains** es el desarrollador y principal patrocinador de Kotlin.

## Características interesantes

- No tiene ;
- Null-safety
- Interoperabilidad con Java
- Claro y conciso [2]

# Sintaxis y ejemplos

---

# Hello World

- Hello world!

---

```
1 fun main(args: Array<String>) {  
2     println("Hello world!")  
3 }
```



[3]



# Variables y constantes

- **var** para variables, **val** para constantes
- Inferencia de tipo

---

```
1 val age = 22
2
3 // age = 100 ⇒ Error (age is constant)
4 // age = "Hola" ⇒ Error (type mismatch)
```

---



- Especificar el tipo

---

```
1 val age: Int = 22
2 val height: Double = 170.2
```

---



# Variables y constantes

```
1  var age = 22
2  age = 90 // OK, age variable & Int
3
4  val name: String
5
6  if (age == 22) {
7      name = "Daniel"
8  } else {
9      name = " Name here ".trim().toLowerCase()
10 }
11 // or
12 val name: String =
13     if (age == 22) {
14         "Daniel"
15     } else {
16         " Name here 33 ".trim().toLowerCase()
17     }
```



- Instanciar clases

---

```
1 val daniel = Person(name = "Daniel", age = 22)
2 daniel.run(distance = 100.k + 890)
```

---



# Todo es un objeto

- Los literales, sean numéricos o texto, son objetos de clases que pueden directamente invocar métodos.

---

```
1 1.rangeTo(30) // or: 1..30 // [1, 30]
2 30.6.hashCode()
3 "TeSt".toLowerCase()
4 9.div(2.0) // 4.5
5 9 / 2 // 4
```

---



- Declaración habitual

---

```
1 fun factorial(n: Long): Long {  
2     if (n ≤ 1) {  
3         return 1  
4     }  
5     return n * factorial(n-1)  
6 }
```

---



- Declaración acortada

---

```
1 fun factorial(n: Long): Long =  
2     if (n ≤ 1) 1  
3     else n * factorial(n-1)
```

---



Declaración de clase con:

- Constructor
- Propiedades
- toString()
- equals(), hashCode()
- copy(), etc

---

```
1 data class Person(val name: String, var age: Int) {  
2     fun run(distance: Long) = println("Running $distance m")  
3 }
```

---



# Classes - Java

---

```
1  import kotlin.Metadata;
2  import kotlin.jvm.internal.Intrinsics;
3  import org.jetbrains.annotations.NotNull;
4  import org.jetbrains.annotations.Nullable;
5
6  @Metadata(
7      /* ... */
8  )
9  public final class Person {
10     @NotNull
11     private final String name;
12     private int age;
13
14     public final void run(long distance) {
15         String var3 = "Running " + distance + " m";
16         System.out.println(var3);
17     }
18
19     @NotNull
20     public final String getName() {
21         return this.name;
22     }
23
24     public final int getAge() {
25         return this.age;
26     }
27 }
```

---

```
28     public final void setAge(int var1) {
        this.age = var1;
    }

30
32     public Person(@NotNull String name, int age) {
        super();
        Intrinsic.checkParameterIsNotNull(name, "name");
34         this.name = name;
        this.age = age;
36     }

38     @NotNull
    public final String component1() {
40         return this.name;
    }

42
44     public final int component2() {
        return this.age;
    }

46
48     @NotNull
    public final Person copy(@NotNull String name, int age) {
        Intrinsic.checkParameterIsNotNull(name, "name");
50         return new Person(name, age);
    }
```

---



---

```
52 @NotNull
   public static Person copy$default(Person var0, String var1, int var2, int var3,
   Object var4) {
54     if ((var3 & 1) != 0) {
   var1 = var0.name;
56     }
   if ((var3 & 2) != 0) {
58     var2 = var0.age;
   }
60     return var0.copy(var1, var2);
   }

62
   @NotNull
64   public String toString() {
   return "Person(name=" + this.name + ", age=" + this.age + ")";
66   }

68   public int hashCode() {
   return (this.name != null ? this.name.hashCode() : 0) * 31 + this.age;
70   }
```

---

---

```
70  public boolean equals(@Nullable Object var1) {  
71      if (this != var1) {  
72          if (var1 instanceof Person) {  
73              Person var2 = (Person)var1;  
74              if (Intrinsics.areEqual(this.name, var2.name) && this.age == var2.age) {  
75                  return true;  
76              }  
77          }  
78          return false;  
79      } else {  
80          return true;  
81      }  
82  }  
}
```

---

# Classes - Polimorfismo

```
1 open class Shape
2
3 open class Polygon(val sidesAndLength: Map<Int, Int>): Shape() {
4     val sides = this.sidesAndLength.size
5 }
6
7 open class Rectangle(
8     val horizontalLength: Int,
9     val verticalLength: Int
10 ): Polygon(sidesAndLength =
11     (1..4).map {
12         it to if (it % 2 == 0) horizontalLength else
13             verticalLength
14     }.toMap()
15 )
16 data class Square(val sidesLength: Int):
17     Rectangle(sidesLength, sidesLength)
```



# Classes - Polimorfismo

---

```
1 val square = Square(sidesLength = 90)
2 println(square.sides)
3
4 val rectangle = square as Rectangle
5 println(rectangle.horizontalLength)
6
7 println("First side: ${square.sidesAndLength[1]}")
8 println(square.sidesAndLength) // {1=90, 2=90, 3=90, 4=90}
```

---



# Enum class & when

```
1 enum class Color(val value: String) {  
2     RED("FF0000"), GREEN("00FF00"), BLUE("0000FF")  
3 }  
4  
5 fun main(args: Array<String>) {  
6  
7     val myColor = Color.RED  
8  
9     when (myColor) {  
10         Color.RED → println(Color.RED.value)  
11         Color.GREEN, Color.BLUE → println("Green or blue")  
12     }  
13 }
```



# when con rangos

```
1 val number = 100
2
3 when (number) {
4     in 0..50 → println(" ... ")
5     !in 80..200 → println(" ... ")
6     in 100..200 → println("yes")
7     else → {
8         println("nope")
9     }
10 }
```



# For, Map, Range, List

```
1 val names = listOf("Daniel", "John", "Jack")
2 for (name in names) {
3     println(name)
4 }
5
6 val nameAgeMap = mapOf("Daniel" to 22, "Michael" to 44)
7 println(nameAgeMap) // {Daniel=22, Michael=44}
8
9 for (i in 0 until 100) { /* ... */ }
10 for (i in 0..100) { /* ... */ }
11 for (i in 1..15 step 2) { /* ... */ }
12 for (i in 10 downTo 0) { /* ... */ }
13 //
14 val (name, age) = Person(name = "Daniel", age = 22)
15 Random.nextInt(0, 10) // kotlin.random
```



# Interoperabilidad

---



# Kotlin en Java

---

```
1 import org.jetbrains.annotations.NotNull;
2 import java.util.List;
3
4 public class People {
5
6     private final List<Person> list;
7
8     public List<Person> getList() {
9         return this.list;
10    }
11
12    People(@NotNull List<Person> people) {
13        this.list = people;
14        this.list.get(0).getName();
15    }
16 }
```

---

---

```
1 val danielAndJohn = listOf(  
2     Person(name = "Daniel", age = 22),  
3     Person(name = "John", age = 30)  
4 )  
5 val people = People(danielAndJohn) // people = ... NOT allowed  
6 println(people.list[0].name)
```

---

# Avanzado

---

# Optional

- Para tipos que pueden ser nulos, usamos tipos opcionales, representados con el símbolo ?

---

```
1 var abc: String = "abc"
2 abc = null // compilation error
3
4 val daniel = Person(name = "Daniel", age = 22)
5 val john = Person(name = "John", age = 44)
6
7 val idPerson = mapOf(0 to daniel, 10 to john)
8 println(idPerson[0]?.name ?: "unknown") // Elvis operator
9
10 // optional value
11 var name: Person? = idPerson[0]
12 name = null // oK
```

---



# Extensions

- Cuando queramos añadir funcionalidades a clases existentes sin herencia.

```
1  enum class Color(val rgb: Int) {  
2      RED(0xFF0000), GREEN(0x00FF00), BLUE(0x0000FF);  
3      val hexString = this.rgb.toHexString()  
4  }  
5  
6  fun Int.toHexString(): String {  
7      return java.lang.Integer.toHexString(this)  
8  }  
9  
10 fun String.toColor() = Color.valueOf(this.toUpperCase())  
11  
12 val Int.M: Long  
13     get() {  
14         return this.toLong() * 1_000_000  
15     }  
16 val Int.k get() = this.toLong() * 1_000
```



# Sobrecarga de operadores


- Para añadir operadores

```
1 operator fun String.times(n: Int): String {  
2     return (0 until n).joinToString(separator = "") { this }  
3 }  
4  
5 operator fun String?.not(): Boolean {  
6     return this == null || this.isEmpty()  
7 }  
8  
9 fun main(args: Array<String>) {  
10     println("Hello" * 3) // HelloHelloHello  
11     val name = " ".trim()  
12     if (!name) {  
13         /* ... */  
14     }  
15 }
```




# Lambdas & programación funcional

```
1 fun calculate(a: Int, b: Int, operation: (Int, Int) → Int):  
    Int {  
2     return operation(a, b)  
3 }  
4  
5 val result = calculate(10, 20) { a, b → a * b }
```



```
1 val sumOfLargeNamesLength =  
2     listOf("Daniel", "John", "Juan!")  
3         .filter { it.length > 4 }  
4         .map { it.length }  
5         .reduce { accumulated, next → accumulated + next }  
6  
7 println(sumOfLargeNamesLength) // 11
```



# Coroutines

- Operaciones en paralelo de forma asíncrona.

```
1 import kotlinx.coroutines.*
2 import kotlin.system.measureTimeMillis
3
4 suspend fun doingSomethingHeavy(): Int {
5     delay(1000L)
6     return 42
7 }
8 suspend fun doingSomethingHeavy2(): Int { /* */ }
9
10 fun main() = runBlocking {
11     val time = measureTimeMillis {
12         val result1 = async { doingSomethingHeavy() }
13         val result2 = async { doingSomethingHeavy2() }
14         println("Results: ${result1.await() +
15                 result2.await()}") // 462
16     }
17     println("Completed in: ${time.div(1000)}s") // 1s
18 }
```





# Bibliografía

---

## Bibliografía i



Kotlin 1.0 release - jetbrains blog.

**URL: <https://bit.ly/2dXuodq>.**



Kotlin - github.

**URL: <https://github.com/JetBrains/kotlin>.**



Kotlin - reference.

**URL: <https://kotlinlang.org>.**

### Recomendaciones:

- <https://kotlinlang.org>
- <http://devdocs.io>
- <https://zealdocs.org>
- <https://kotlinlang.org/community/>
- <https://github.com/illescasDaniel/Kotlin-Slides>