



# QUITECTURA Y CICLO DE VIDA DE COMPONENTES ANDROID



## ¿QUÉ VAMOS A VER?

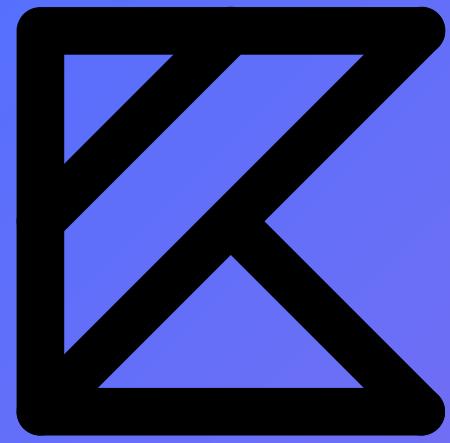
- Kotlin.
- Nuevo Empty Activity.
- Kotlin en consola.
- Variables
- Null Safety
- Cambiar tipo de dato.
- Concatenaciones.
- Try/Catch





# ¡VAMOS A COMENZAR!



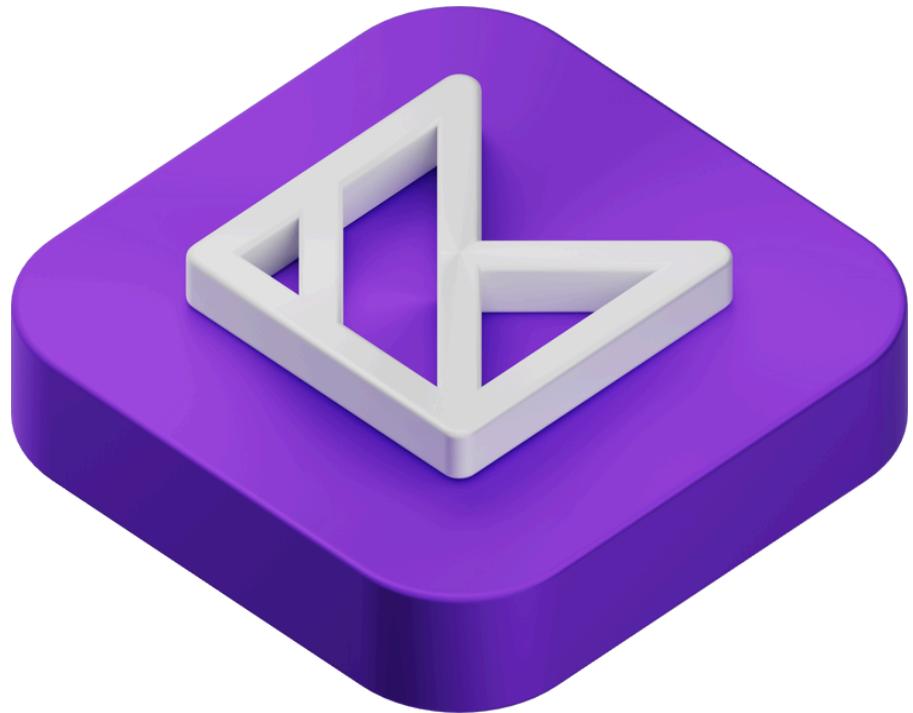


# KOTLIN

# KOTLIN



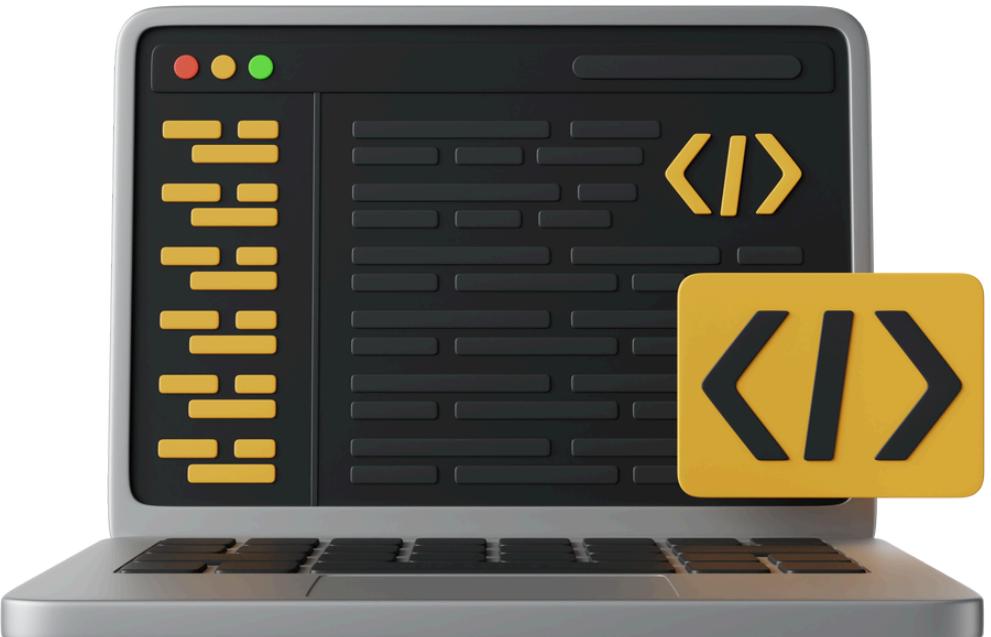
- Es lenguaje de programación de tipado estático, 100% interoperable con Java, creado por la empresa Jetbrains.
- **¿Qué lo hace especial?**
- **Multiplataforma:** Genera archivos de tipo bycode que le permiten correr en la JVM, convirtiéndose así en un lenguaje multiplataforma.
- **Conciso:** A comparación de Java, es un 40% menos de código que se debe escribir.
- **NullSafety:** Nos permite tener la seguridad al trabajar de que no tendremos un NullPointerException. Podemos incluso asignar un valor por defecto para darle mayor seguridad.





# VENTAJAS DE KOTLIN

- **Es seguro contra nulos:** En Java tenemos una cantidad enorme de errores que arrojan *NullPointerException*, pero en Kotlin nos vamos a olvidar de este tipo de errores.
- **Ahorra código:** Escribiremos menos líneas que antes, y con una sintaxis mucho más simple.
- **Características funcionales:** Podemos decidir si vamos a trabajar bajo el paradigma de la *programación orientada a objetos*, o la *programación funcional* (podemos incluso mezclarlos). Esto nos dará mayor libertad a la hora de crear la lógica de nuestros proyectos.



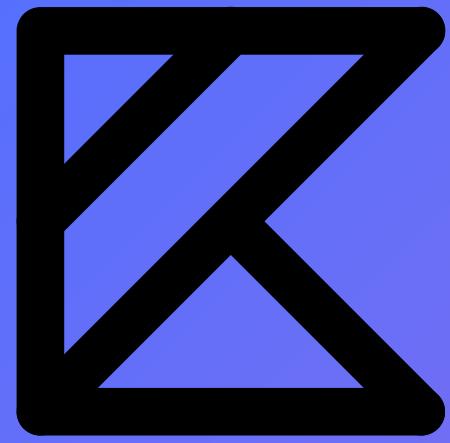


# VENTAJAS DE KOTLIN

- **Fácil de usar:** Su curva de aprendizaje es mucho más sencilla que la de otros lenguajes de programación, sobre todo cuando nos mudamos de un lenguaje de tipado fuerte como lo es Java.
- **Es el momento:** En 2017 Kotlin fue soportado por Android como una alternativa novedosa. En 2019, Google lo presenta como el lenguaje oficial de Android, y en 2024 nuevamente Google apuesta todas sus fichas a Kotlin y su nueva modalidad de multiplataforma. La verdad es que no existe mejor época para dedicarse profesionalmente a este lenguaje de programación.



Kotlin

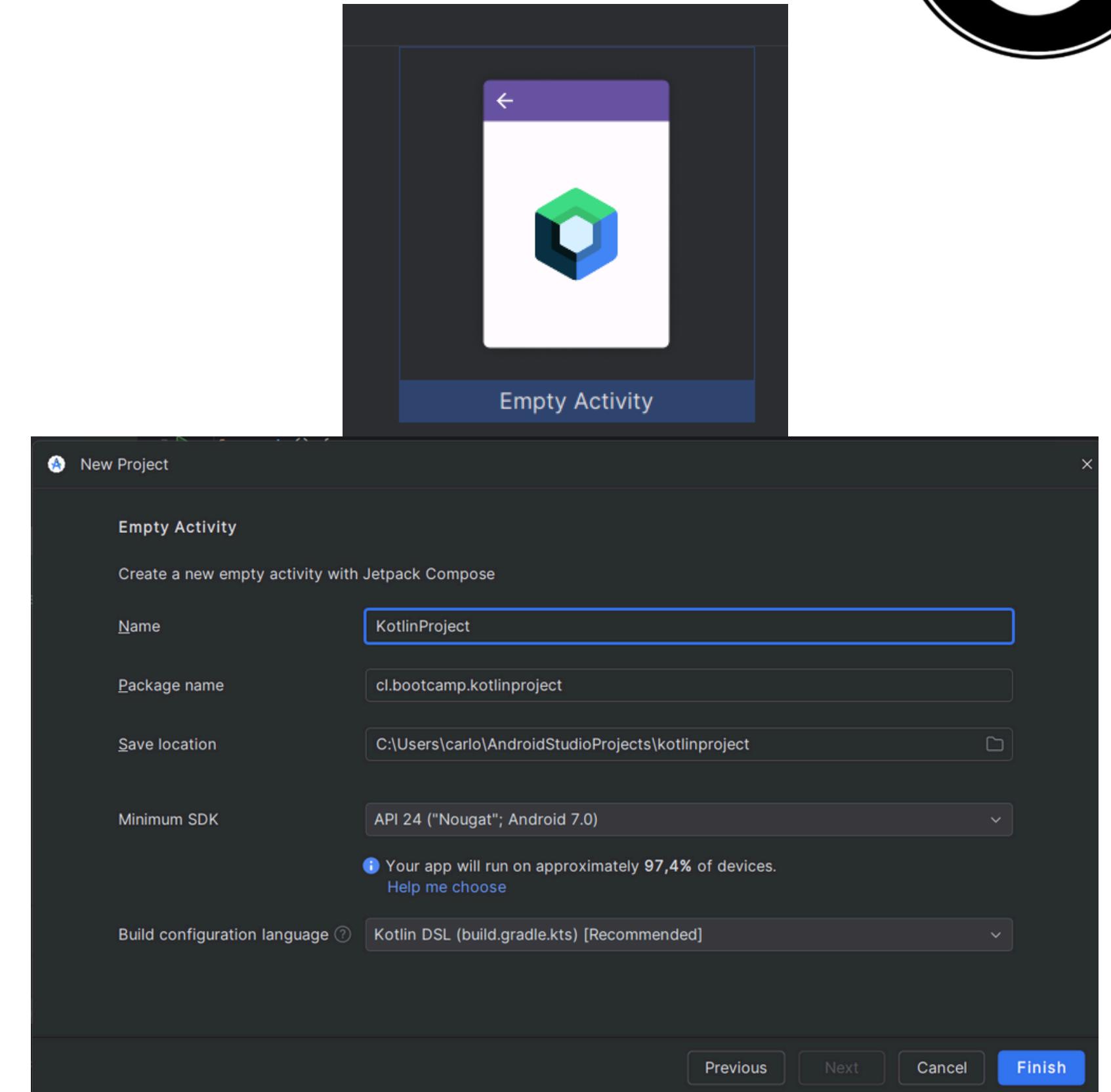


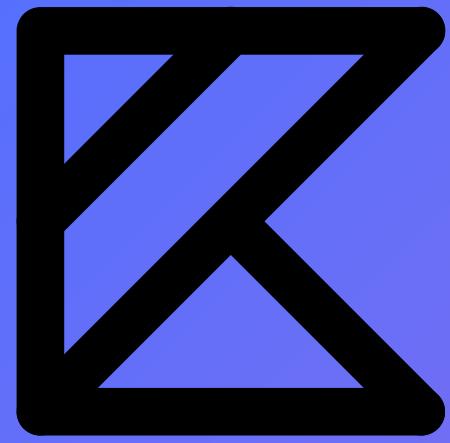
# NUEVO EMPTY ACTIVITY



# EMPTY ACTIVITY

- Al momento de crear un nuevo proyecto, tenemos múltiples opciones para inicializarlo, siendo **Empty Activity** la primera que Android Studio nos recomienda.
- Esta vez la vamos a seleccionar y nos daremos cuenta que es muy similar a *Empty Views Activity*, con la salvedad de que no nos da la opción de seleccionar un lenguaje de programación, eso porque automáticamente va a reconocer a Kotlin como el lenguaje de nuestro proyecto.





# KOTLIN EN CONSOLA



# FUN MAIN()

- Al crear el proyecto nos vamos a encontrar con un mundo totalmente diferente al que nos presentaba el Activity anterior, pero no se asusten, tendremos tiempo para conocer sus componentes y entender el funcionamiento de **Jetpack Compose**.
- De entrada no tocaremos nada del *MainActivity.kt* que nos generó, sino que daremos segundo clic sobre la carpeta que lo contiene y crearemos nuestro propio **File** para trabajar los fundamentos de Kotlin en consola.
- En nuestro nuevo .kt crearemos una función llamada **fun main () {}**

```
Android <--> MainActivity.kt x
  app
    manifests
    kotlin+java
      cl > New
        Java Class
        Kotlin Class/File (selected)
        Android Resource File
        Android Resource Directory
      Add C++ to Module

New Kotlin Class/File
  Main
    Class
    File (selected)
    Interface
    Data class
    Enum class
    Annotation
    Kotlin script
    Kotlin worksheet
    Object

Main.kt x
  package cl.bootcamp.kotlinproject
  fun main() { }
```



# HOLA MUNDO EN KOTLIN

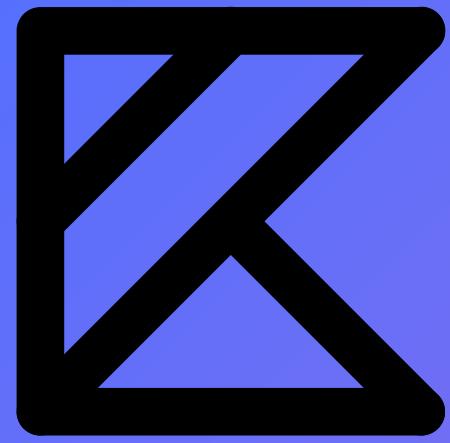
- Podemos escribir tranquilamente la palabra **main** y apretar tabulador para que nos genere automáticamente la función.
- Ahora vamos a escribir **println()** y dentro ejecutaremos nuestro primer “**Hola Mundo!**”
- **Importante:** En Kotlin no es necesario agregar el *punto y coma* al terminar una línea, aunque tampoco es malo si lo utilizan, de ahora en adelante es a libre elección.

The screenshot shows an IDE interface with the following elements:

- Main.kt File Content:** The code is as follows:

```
1 package cl.bootcamp.kotlinproject
2
3 fun main() {
4     println("Hola Mundo!")
5 }
```
- Run Configuration Menu:** A context menu is open over the code, showing options:
  - Run 'MainKt' Ctrl+Mayús+F10
  - Debug 'MainKt'
  - Run 'MainKt' with Coverage
  - Modify Run Configuration...
- Run Tab:** Shows the current configuration: "MainKt". Below it, the output window displays the result of the execution:

```
"C:\Program Files
Hola Mundo!"
```



# VARIABLES

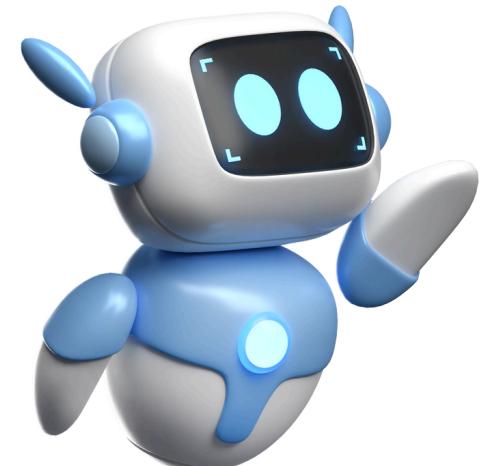


# VARIABLES

- Los tipos de datos que soportan las variables de Kotlin no van a diferir de lo que ya aprendimos en los fundamentos de la programación.
- **String, Int, Double, Float, Boolean**, etc... no van a perder importancia, pero si cambiará la forma en que las vamos a declarar.
- Ya no comenzamos declarando el tipo de dato como en Java, ahora comenzaremos con la palabra reservada **var**, a la que agregaremos el nombre y al final, luego de dos puntos, el tipo de dato que almacenaremos, es más, podríamos ni siquiera agregar el tipo de dato y dejar que Kotlin se encargue de reconocerlo.

```
var num : Int = 10  
var texto : String = "Hola mundo"
```

```
var num2 = 20  
var texto2 = "Hola Kotlin"
```

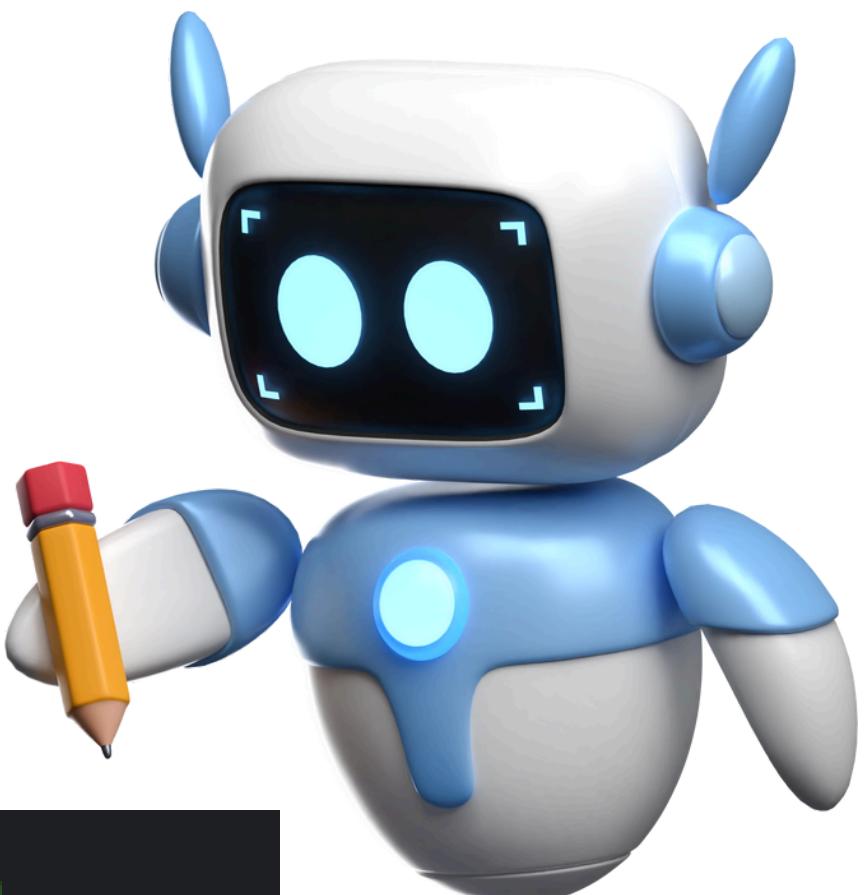


```
println(num2)  
println(texto + num)
```



# CONSTANTES

- En Java vimos que existía la forma de declarar un dato como “constante”, pero no era algo muy utilizado... ahora en Kotlin ocurre lo contrario, con normalidad van a utilizar muchas más constantes que variables.
- Se declaran con la palabra reservada **val** y su funcionamiento es exactamente el mismo de una variable, con la salvedad de que una vez creada ya no se puede modificar.



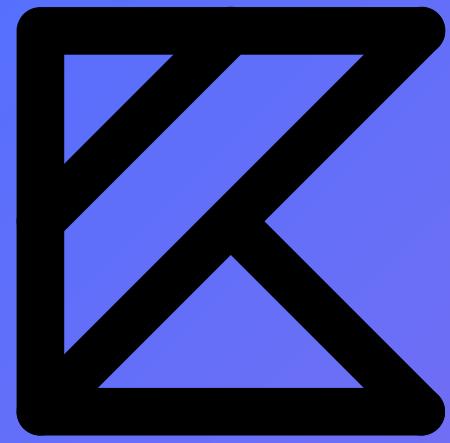
```
val num3 : Int = 50  
val texto4 = "Constante"
```



# ¿VAR O VAL?

- Realmente ninguna es mejor que otra, ambas tienen un funcionamiento específico dentro de cada proyecto y su uso va a depender netamente de cada programador.
- Ahora, es verdad que val puede ser mucho más precisa y funcional, imagínense a **var** como *una mochila enorme* que soporta dentro de ella cualquier cosa... eventualmente es más pesada, por el contrario, **val** sería *un pequeño bolsito transparente* que almacena un objeto y no se vuelve a cambiar su contenido.





# NULL SAFETY



# VALORES OPCIONALES

- ¿Recuerdan que Kotlin es considerado un lenguaje más seguro respecto a datos nulos?
- Null Safety va a ser la herramienta ideal que tendremos para controlar los datos nulos, sobre todo cuando no sabemos si alguna de nuestras variables va a recibir o no datos.
- Vamos a declarar una variable como “nula” con una sintaxis especial, no basta simplemente con indicarle que su campo desde un inicio va a estar nulo.
- ? -> nos dice de que existe la posibilidad de que se almacene un *String* en la variable, pero no hay forma de saber si será así y mucho menos cuándo será.



```
var y : String? = null
```

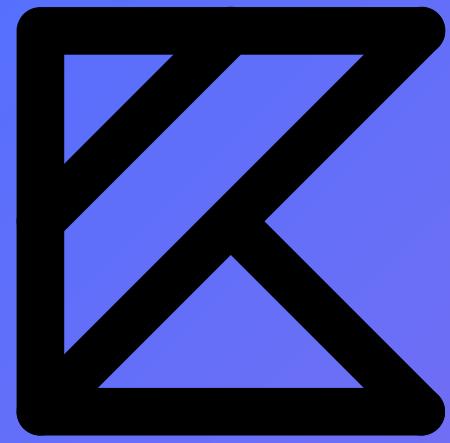


# VALORES OPCIONALES

- También vamos a tener la forma de saber si un momento dado de nuestro código, la variable está almacenando un dato nulo o no.
- Para eso utilizaremos **.let**
- En caso de que el dato no sea nulo, se ejecutará un bloque de código, de lo contrario, no pasará nada

```
var y : String? = null  
  
y = "Hola"  
  
y?.let {  
    println("tiene valor")  
}
```





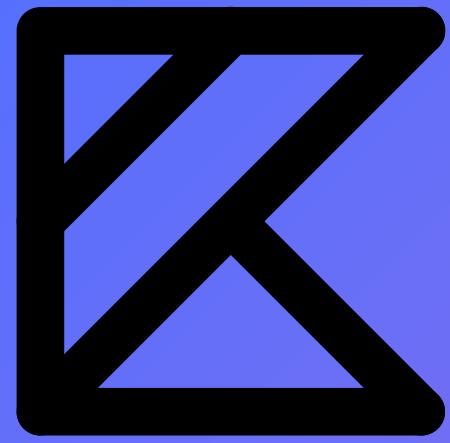
# CAMBIAR TIPO DE DATO



# CAMBIAR TIPO DE DATO

- Nada nuevo para nosotros, pero algo muy común dentro de los diferentes proyectos que realizamos.
- En Kotlin simplemente tendremos que recurrir a una función en específico para cambiar al tipo de dato que deseamos.
- **.toInt()** = Convierte un dato a tipo Int.
- **.toString()** = Convierte un dato a tipo String
- **.toDouble()** = Convierte un dato a tipo Double.
- **.toBoolean()** = Convierte un dato a tipo Booleano.
- Estos son algunos de los más comunes, en la realidad nos vamos a encontrar con un montón de métodos para diferentes tipos de datos que nos podemos encontrar.

```
val num1 = 5
val num2 = "10"
val res = num1 + num2.toInt()
println(res)
```



# CONCATENACIONES



# CONCATENACIONES

- A estas alturas ya no nos escaparemos de la forma clásica de concatenar, pero es cierto de que Kotlin cuenta con otras formas mucho más llamativas.

- Concatenar con **+**

```
val t1 = "Hola"  
val t2 = "Mundo"  
val texto = t1 + " " + t2  
println(texto)
```



- Concatenar con **.plus()**

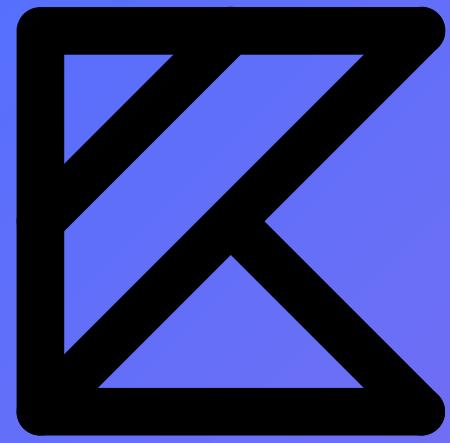
```
val resultado = t1.plus(" ").plus(t2)  
println(resultado)
```

- Concatenar con **.joinToString()**

```
val array = arrayOf("Hola", "Mundo")  
val res2 = array.joinToString(" ")  
println(res2)
```

- Concatenar con **\$**

```
println("Soy el número: $num1")
```



# ESCRIBIR EN CONSOLA



# ESCRIBIR EN CONSOLA

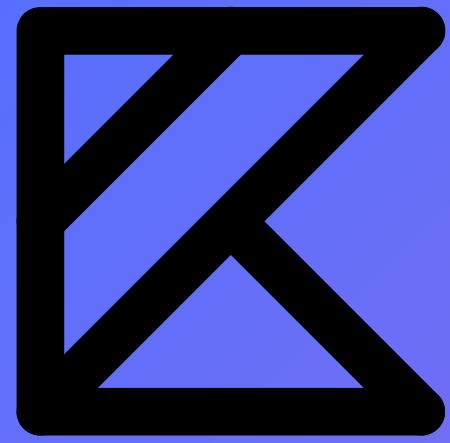
- ¿ Scanner input = new Scanner(System.in) ?
- Por suerte **no**, esta vez no vamos a necesitar recurrir a la creación de un objeto, ni tampoco a importar una librería externa que nos ayude a hacerlo, el mismo Kotlin tiene su propio método que nos facilitará el trabajo.
- **readIn()**: Este método se encargará de crear la instancia para ingresar un dato por consola y almacenarlo dentro de un variable.
- Por defecto, el método readIn solo almacena datos de tipo String, pero si agregamos la conversión de dato al mismo método, eso puede cambiar.

```
println("Ingresa tu nombre")
val nombre = readln()
println(nombre)
```

```
println("Escribe el primer valor")
var n1 = readln().toInt()

println("Escribe el segundo valor")
var n2 = readln().toInt()

var suma = n1 + n2
println("Suma: $suma")
```



# TRY / CATCH



# TRY / CATCH

- Aprovechando que acabamos de simular una interacción primitiva en consola, nunca vamos a saber lo que al usuario se le puede ocurrir ingresar, es por eso que debemos prevenir algún error y capturarlo antes de que “rompa” nuestra aplicación.
- Con **Try / Catch** tendremos dos bloques, siendo el primero (**try**) en dónde se va a ejecutar el código sin problema, y el segundo (**catch**) en dónde podremos mandar algún mensaje con el error o mostrarlo directamente para enterarnos de lo que está ocurriendo.

```
try {  
    println("Escribe el primer valor")  
    var n1 = readln().toInt()  
  
    println("Escribe el segundo valor")  
    var n2 = readln().toInt()  
  
    var suma = n1 + n2  
    println("Suma: $suma")  
} catch (e: NumberFormatException) {  
    println("Error: escribe un número válido - $e")  
}
```



**GRACIAS POR  
LA ATENCIÓN**