# 1. Conceptos básicos Kotlin

▼ **Descripción**

**Kotlin** es un lenguaje de programación moderno, expresivo y conciso, que facilita en gran medida el desarrollo no solo de aplicaciones móviles sino de otras plataformas, es interoperable con el lenguaje Java ya que busca brindar una nueva alternativa a este lenguaje sin embargo corre sobre su maquina virtual, adicional es junto con java el lenguaje oficial para desarrollo de aplicaciones nativas para **android**.

Tal como se menciona en la pagina de desarrolladores de **android**, **Kotlin** es un proyecto gratuito y de código abierto registrado bajo la licencia de Apache 2.0, el código del proyecto está publicado en **gitHub** y a cargo principalmente del equipo de desarrollo de **JetBrains** con contribuciones de Google y otros.

**Nota**: Actualmente se tiene una versión **Alpha de Kotlin Multiplatform Mobile (KMM)** lo que permitiría hacer un único desarrollo tanto para **android** como para **iOs**

▼ **Características**

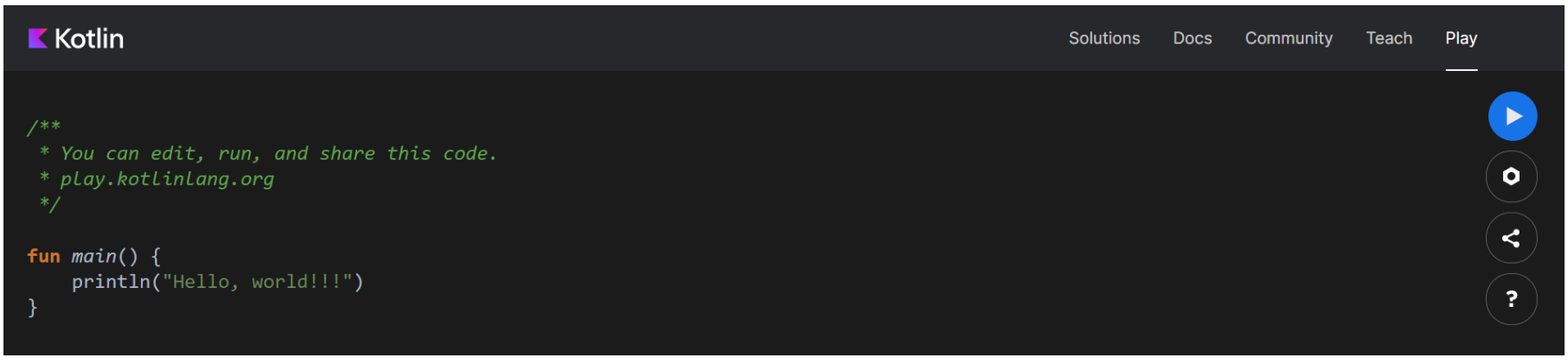
- Fue creado por JetBrains
- Se puede usar en Ciencia de Datos, Desarrollo Web, Desarrollo Móvil entre otros.
- Lenguaje de Alto Nivel
- Multiplataforma
- Sintaxis Simple
- Es orientado a Objetos
- Soporta null safety
- Lenguaje Oficial para desarrollo de Apps Android
- Interoperable con java
- No es obligatorio el punto y coma (;)

▼ **Primeros Pasos**

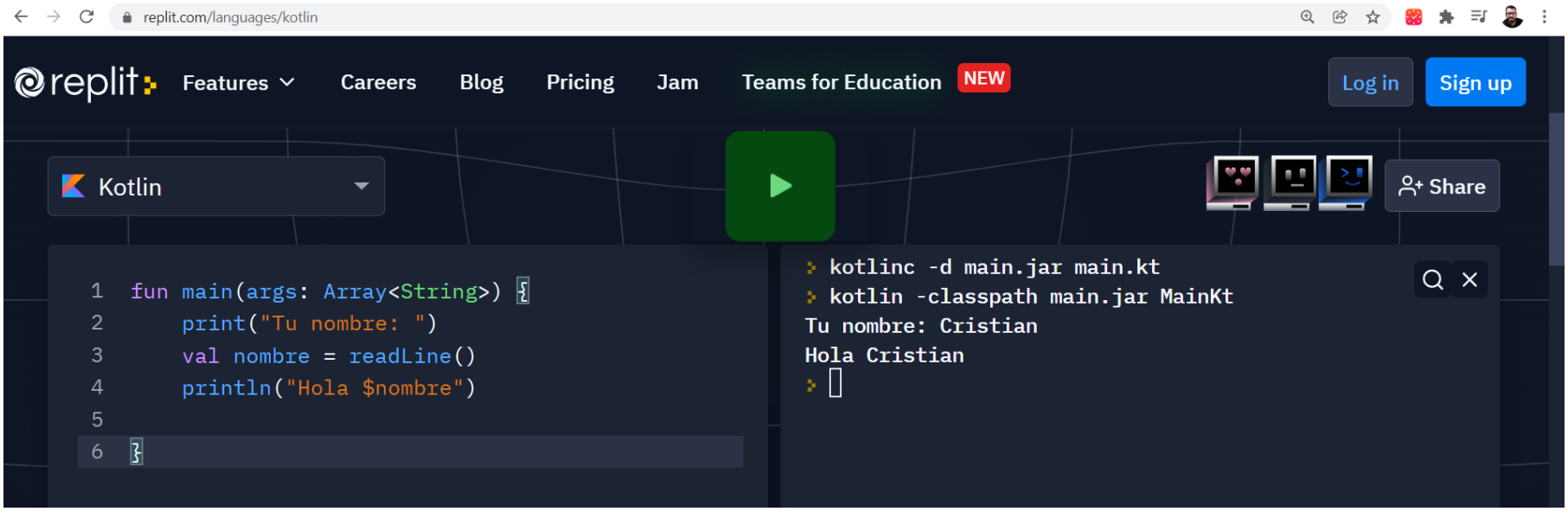
Para trabajar con Kotlin no realizaremos el proceso de instalación como estamos acostumbrados al hacerlo con otros lenguajes de programación, al contrario vamos a usar herramientas que brindan ya el entorno necesario para el trabajo con este lenguaje, estos entornos pueden ser IDE’s como IntelliJidea, Android Studio o trabajar directamente en los editores online que la pagina oficial brinda (estos últimos mientras se aprende lo básico del lenguaje) o los disponibles en internet.

Para iniciar con el lenguaje se recomienda revisar la página oficial https://kotlinlang.org/

desde allí se puede acceder a el editor online que nos permitirá realizar algunas prácticas básicas. https://play.kotlinlang.org/

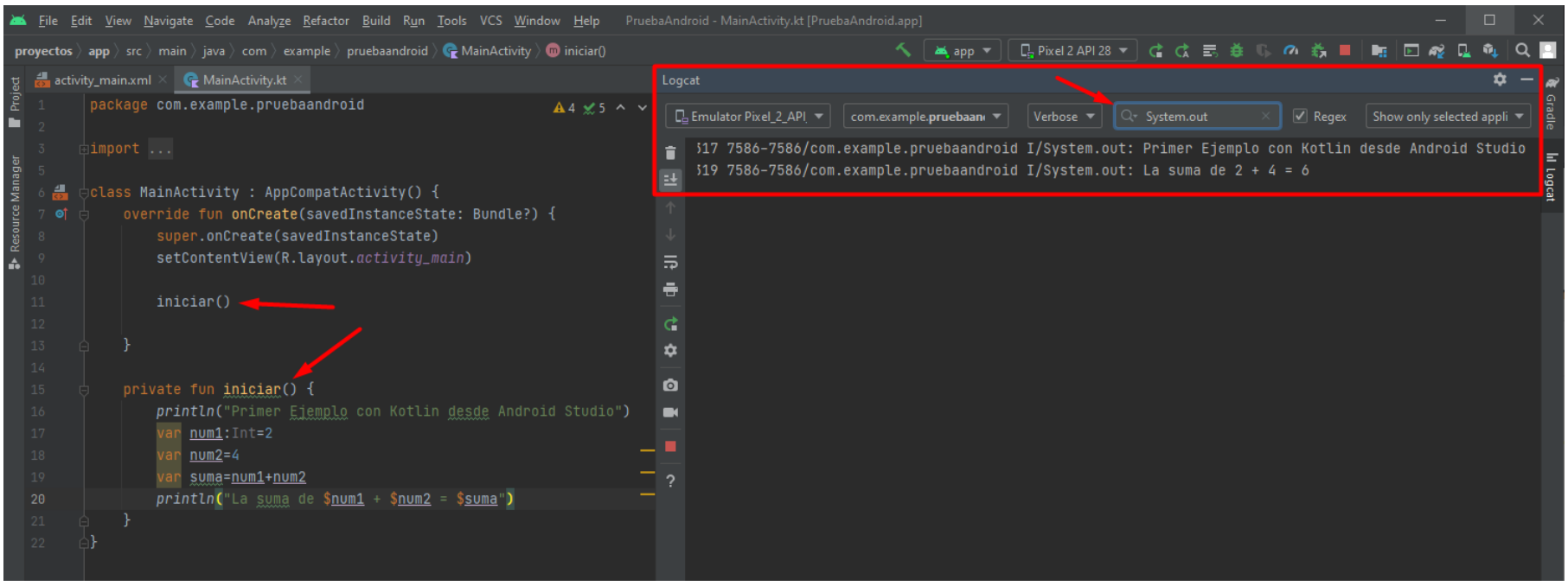



Como se menciona no es la única herramienta online que puede utilizarse, se puede probar otras como https://replit.com/languages/kotlin para realizar las prácticas iniciales




Nótese que se trabaja directamente sobre la función **main**, en la segunda imagen se usan parámetros que permiten trabajar con entrada y salida de datos, esto lo veremos más adelante en este documento.

También se puede usar la herramienta Android Studio y filtrando los mensajes que pongamos en consola.




**Nota:** Se recomienda también seguir la documentación oficial en https://kotlinlang.org/docs/home.html como complemento a esta guía.

### Video de Apoyo: ¿Como empezar con Kotlin?

https://www.youtube.com/watch?v=HEkc00WunNs

### ▼ Comentarios de Código

Los comentarios de código son bloques de texto con los que podemos comentar nuestros algoritmos, esto ofrece a las personas que lean nuestro código mayor información.

Como **Kotlin** tiene inspiración en diferentes lenguajes, entre ellos **java**, en este caso se comparte la misma sintaxis para los comentarios de código, de esta manera normalmente tenemos comentarios de una sola línea que nos dan información puntual y multilínea que nos detallan mucho más un proceso.

```
//Esto es un comentario de una sola linea
//Se declara la variable edad
edad=15

/*
*Esto es un comentario multilinea, usado cuando
*debemos explicar un proceso más en detalle
*/
```

## ▼ Variables, constantes, Tipos de Datos.

En **Kotlin** todo es un objeto por esa razón se puede acceder a funciones y propiedades en cualquier variable, por esta razón los diferentes tipos de datos pueden ser representados en tiempo de ejecución como valores primitivos de los que conocemos de lenguajes como java, pero al momento de desarrollar se definen como clases normales según su sintaxis, veamos un pequeño repaso.

### ▼ Variables mutables e inmutables

**Kotlin** define el concepto de **variables mutables e inmutables** para referirse a las **variables** y **constantes** como las conocemos, es decir una **variable mutable** puede cambiar el dato almacenado durante la ejecución del programa y una **variable inmutable** significa que cuando le asignamos un valor no puede cambiar más a lo largo del programa.

### ▼ Variables mutables

Una variable es un contenedor que puede almacenar información y puede cambiar en el tiempo pues su contenido puede variar, básicamente se puede definir como un nombre que identifica una dirección de memoria con un tipo de dato especifico.

Las **variables (mutables)** en **Kotlin** se pueden crear usando la palabra reservada **var** y asignando un valor a un nombre definiendo su tipo de dato (ver Ej1).

o también se puede hacer uso de la inferencia de tipos que permite crear una variable sin necesidad de definir el tipo de dato o realizar previamente una declaración (ver Ej2), esto es algo similar a como lo hacen lenguajes como python y en el que el compilador **infiere** de forma automática el tipo de dato al que pertenece.

**Ej1.** var nombreVariable:TipoDeDato= valor
**Ej2.** var nombreVariable= valor

### ▼ Constantes

Las constantes corresponden a datos que una vez inicializadas no pueden cambiar su valor, en kotlin estas son definidas usando la palabra reservada val, como recomendación se deberían usar las constantes en mayúsculas por organización.

Las **constantes (variables inmutables)** en **Kotlin** se pueden crear usando la palabra reservada **val** y se definen igual a como se crean las variables mutables, la única diferencia es que no se puede reasignar su valor.

**Ej1.** val nombreConstante:TipoDeDato= valor
**Ej2.** val nombreVariable= valor

### ▼ Tipos de datos

Ya vimos que las variables corresponden a contenedores de memoria donde se almacenarán valores, y esos valores deben estar asociados a un tipo de dato especifico, en general se tienen los siguientes tipos de datos:

**Numéricos** (enteros, decimales)
**Textos** (un carácter o cadena de caracteres)
**Lógicos** (verdadero o falso)

**Kotlin** define una serie de tipos integrados que representan números tanto enteros como decimales

**Enteros.**

| Type | Size (bits) | Min value | Max value |
|---|---|---|---|
| `Byte` | 8 | -128 | 127 |
| `Short` | 16 | -32768 | 32767 |
| `Int` | 32 | -2,147,483,648 ($-2^{31}$) | 2,147,483,647 ($2^{31} - 1$) |
| `Long` | 64 | -9,223,372,036,854,775,808 ($-2^{63}$) | 9,223,372,036,854,775,807 ($2^{63} - 1$) |

Si se hace inferencia de tipos al momento de declarar una variable y el valor inicial excede el valor del tipo, entonces se declarará como el tipo siguiente, veamos:

```
var num1=100 //Por inferencia de tipos y estar en el rango, num1 es Int
var num2=129 //Por inferencia de tipos y estar en el rango, num2 es Short
```

**Decimales o coma flotante.**

| Type | Size (bits) | Significant bits | Exponent bits | Decimal digits |
|---|---|---|---|---|
| `Float` | 32 | 24 | 8 | 6-7 |
| `Double` | 64 | 53 | 11 | 15-16 |

Igual que en java cuando trabajamos con variables flotantes se debe agregar el sufijo f o F

```
val a = 2.7 // Double
val a1:Double = 2.7 // Double
val b = 2.7f //Float
val b1:Float = 2.7f //Float
```

**Textos y caracteres.**

Para los datos de texto se maneja la clase **String** que permite almacenar cadenas de caracteres mientras que para un solo carácter se utiliza **Char** y su forma de usarlo es igual a como se trabaja en java usando comillas dobles y comillas simples respectivamente.

Los caracteres especiales comienzan con una barra invertida de escape `\`. Las siguientes secuencias de escape son compatibles: `\t`, `\b`, `\n`, `\r`, `\'`, `\"`, `\\` y `\$`.

```
var nombres="Cristian David" //con inferencia de tipos
var apellidos:String="Henao Hoyos"
var inicialNombre1='C' //con inferencia de tipos
var inicialApellido1:Char='H'
```

**Lógicos o Booleanos.**

Los Booleanos se definen con la palabra Boolean y representa objetos lógicos que pueden tener dos valores “true y false” adicional puede también tener una contraparte anulable al declararse Boolean? lo que permite asignar el valor de null

```
var verdadero:Boolean = true
var falso:Boolean = false
var nulo:Boolean? = null
```

**Conversiones o Casting**

En kotlin como en muchos lenguajes de programación, los tipos de datos más pequeños no son subtipos de los más grandes, esto significa que al momento de asignar un valor de tipo de una variable Byte a una variable Int requiere de una conversión explicita.

```
val b: Byte = 1
// val i: Int = b // ERROR
val i1: Int = b.toInt() // se hace el casting convirtiendo el valor a entero

//Todos los tipos de números admiten conversiones a otros tipos:
toByte(): Byte
toShort(): Short
toInt(): Int
toLong(): Long
toFloat(): Float
toDouble(): Double
toChar(): Char
```

```
//De la misma manera podriamos convertir el valor de un texto en un dato número con las funciones anteriores
var x="2.5".toDouble()  //Obtiene el valor numerico 2.5
```

**¿Cómo nombrar una Variable?**

Los identificadores representan la forma correcta de definir nombres de variables, para esto como estándar se debe tener en cuenta lo siguiente:

- El primer carácter debe ser un caracter alfabético (a…z, A…Z) o $, _
- Después del primer caracter pueden ir caracteres alfanuméricos (a…z, A…Z) o $, _, (0…9)
- Los identificadores no pueden ser palabras reservadas del lenguaje
- Se recomienda aplicar la regla camelCase
- Los identificadores no pueden tener espacios, signos de puntuación, tildes ni otro carácter diferente a los mencionados.

Veamos algunos ejemplos donde se aplican los conceptos mencionados anteriormente.

```
//var : Variables Mutables
//val : Variables Inmutables - Constantes

var numeroA:Byte=12
var numeroC:Short=8
val NUMERO_B:Int=24
var numeroD:Long=12584

var numeroE:Float=2.5f
var numeroF:Double=2.5

var caracter:Char='a'
var nombre:String="Cristian"
var logico:Boolean=true

var edad:Int=numeroA.toInt()
var codigo="CDHH"
```

**Video de Apoyo: Variables y Tipos de Datos en Kotlin.**

https://www.youtube.com/watch?v=YWztyqwUNpE

▼ **Entrada y Salida de Datos**

Los programas operan sobre un conjunto de datos de entrada, generando algún tipo de salida; estos necesitan comunicarse con su entorno, tanto para leer datos e información que deben procesar, como para retornar o mostrar los resultados obtenidos.

Cada lenguaje de programación tiene clases o funciones propias que permiten realizar este proceso de entrada de datos, donde normalmente el valor ingresado se almacena en una variable y la salida corresponde a la impresión de los datos almacenados.

```
fun main() {
  //Imprime un mensaje en consola sin salto de linea
  print("Este es un mensaje de saludo de linea! ")
  //Imprime el mensaje en la misma linea del anterior y
  //luego da salto de linea
  println("Este es un mensaje con salto de linea!")
  //Imprime un mensaje con salto de linea
  println("y el 3er mensaje")

  //Todo lo que se ingrese por teclado es un texto,
  //incluso los números inicialmente son procesados
  //como texto pero luego se deben transformar
  print("Ingrese su nombre: ")
  val nombre = readLine()
  //Si queremos imprimir la información se puede hacer
  //de la forma tradicional concatenando la variable con
```

```
  //el signo +
  println("Hola "+nombre)
  //o se puede usar el signo $ para evitar hacerlo
  println("Bienvenido $nombre")

  //Para el ingreso de datos númericos podemos convertir a entero así:
  print("Ingrese su edad: ")
  val edad = readLine()!!.toInt() //Se hace uso de !! para evitar error en tiempo de ejecución por posibles valores null
  println("Tu nombre es $nombre y tienes $edad años.")
}
```

## ▼ Cadenas sin Formato, Operadores Aritméticos, Lógicos y Relacionales.

### ▼ Uso de String y cadenas sin formato

Como sabemos, las cadenas en Kotlin están representadas por el tipo String, generalmente una cadena es una secuencia de caracteres entre comillas dobles (””)

Ej: `var cadena="Hola Grupo"`

Estas cadenas tienen el mismo comportamiento que podemos encontrar en otros lenguajes de programación, al pertenecer a la clase String, esta brinda una serie de funciones de utilidad que permiten realizar operaciones sobre la misma.

```
//Acciones sobre la clase String
var nombre="Cristian"
println(nombre.toUpperCase())       // CRISTIAN
println(nombre.toLowerCase())       // cristian
println(nombre.last())              // n
println(nombre.lastIndex)           // 7
println(nombre.length)              // 8
println(nombre.equals("Cristian"))  // true
println(nombre.equals("Juan"))      // false
println(nombre.contains("Cris"))    // true
```

#### Concatenación

Como es común cuando queremos presentar información se debe concatenar el contenido de una variable a una cadena, este proceso lo podemos hacer igual que en java utilizando el signo + o también podemos usar una expresión de plantilla iniciando con un signo de dólar ($) y adicional para realizar otras operaciones podemos hacer uso de expresión entre llaves

```
//Forma tradicional
var num=8
println("El valor del numero es : "+num) //El valor del numero es : 8
//Forma expresión de plantilla
println("El valor del numero es : $num") //El valor del numero es : 8

//Expresión entre llaves
println("$num + 2 es ${num+2}")   //8 + 2 es 10
var nombre="Pepe"
println("$nombre.length es ${nombre.length}") //Pepe.length es 4
```

#### Caracteres de Escape y cadenas sin formato

Anteriormente se presentaron los diferentes caracteres de escape, estos permiten hacer acciones de salto de linea, tabulación entre otros, pero adicional podemos hacer uso de cadenas sin formato para  armar mensajes de forma más simple, solamente agregando 3 comillas al inicio y al final del texto que queramos presentar.

```
//Ej. Usando salto de linea
val saludo="Hola esto es un saludo con salto de linea al final/n"

//Ej. Usando cadena sin formato
val num=8
val mensaje="""
Este es un mensaje en una
cadena sin formato
este es el valor de num=$num
"""
println(mensaje)
```

### ▼ Aritméticos

Estos operadores corresponden a los usados para labores académicas cotidianas tales como procesos de sumas, restas, división, multiplicación, modulo, incremento, decremento, su aplicación se puede evidenciar en la siguiente tabla.

```
//Suma              +
var suma=2+3 //Resultado 6
//Resta             -
var resta=2-2 //Resultado 0
//Multiplicación    *
var mult=2*2  //Resultado 4
//División          /
//Se debe tener en cuenta que la dicisión de enteros devuelve siempre un entero,
//en el siguiente ejemplo no se contempla la parte decimal, por lo tanto
//imprime 3
var div=7/2   //resultado 3
//Si queremos contemplar la parte decimal entonces se deberia tener declarados
//los números como decimales o al momento de dividir hacer uso de la
//función toDouble() para indicar al compilador que se espera una respuesta
//decimal, así el resultado será 3.5
var div=7/2.toDouble()   //resultado 3.5
//Módulo            %
mod=8%2   //resultado 0
```

**Video de Apoyo: Operadores Aritméticos en Kotlin.**

https://www.youtube.com/watch?v=j_9VGq1muWs

### ▼ Relacionales

Los operadores relacionales permiten definir la relación entre 2 o más expresiones

La forma de usar e interpretar estos operadores es usando la estructura:

**Expresión1 OperadorRelacional Expresión2**

Donde el resultado de la operación anterior se evalúa en términos de verdadero o falso.

```
_____________________________________
| Operador     |    Descripción  |
_____________________________________
|    <         |    Menor Que    |
|    >         |    Mayor Que    |
|    <=        | Menor igual Que |
|    >=        | Mayor igual Que |
|    ==        |    Igual Que    |
|    !=        |  Diferente de   |
_____________________________________

//Ejemplos
var a=5
var b=4
println(a>b)//true
println(a<b)//false
println(a>=b)//true
println(a<=b)//false
println(a==b)//false
println(a!=b)//true
```

**Video de Apoyo: Operadores relacionales en Kotlin.**

https://www.youtube.com/watch?v=YNlb2Bp-K5o

### ▼ Lógicos

Los operadores lógicos permiten la combinación de condiciones para formar una sola expresión lógica, para estos permiten obtener un resultado lógico al complementarse con los operadores relacionales.

```
_____________________________________________________________________________
|   Operador   |    Descripción  |                    Descripción             |
_____________________________________________________________________________
|              |                 | Al usar este operador, si la condición arroja falso,|
|      !       |   Negación (no) | el operador la convierte en verdadero y viceversa   |
|______________|_________________|_____________________________________________|
|              |                 | Es verdadero solo si los 2 (o Todos) los operando   |
|      &&      |  Conjunción (y) | Son verdadero, caso contrario arroja falso          |
|______________|_________________|_____________________________________________|
|              |                 | Es verdadero cuando al menos un operando es         |
|      ||      |  Disyunción (o) | verdadero                                           |
|______________|_________________|_____________________________________________|

var resp=true
println(resp)

//Operador de Negación
println(!resp)  //false
println(!(2>1)) //false

//Operador &&
println(true && resp && true && true && true)         //true
println(true && resp && true && false && true)        //false
println(true && (!resp) && true && true && true)      //false

//Operador ||
println(false || false || false || false || resp)       //true
println(false || false || false || false || (!resp))    //false
println(false || false || false || false || false)      //false
```

**Video de Apoyo: Operadores lógicos en Kotlin.**

https://www.youtube.com/watch?v=2fv1sOE6Qec

### ▼ Rango de Instancias o de Alcance.

Este concepto nos permitirá definir rangos de elementos para facilitar condiciones donde un valor se encuentra dentro de algún intervalo especifico.

```
//a..b  - representa el intervalo entre los valores a y b
//x in a..b - valida si x se encuentra dentro del intervalo a..b
println(12 in 1..8)  //Esto arroja false, ya que 12 no está en el intervalo de 1 a 8
println(2 in 1..8)  //Esto arroja true, ya que 2 si está en el intervalo de 1 a 8
//También se puede negar la validación
//x !in a.. b - valida si x NO se encuentra dentro del intervalo a..b
println(12 !in 1..8)  //Esto arroja true, ya que efectivamente 12 NO está en el intervalo de 1 a 8
println(2 !in 1..8)  //Esto arroja false, ya que 2 SI está en el intervalo de 1 a 8
```

Los intervalos arrojan resultados lógicos en términos de true o false, por lo tanto estos valores pueden ser almacenados en variables y ser procesados

```
var res=2 in 1..5   //almacena true en res
println("¿2 está en el intervalo 1..5? = $res")
```

Igual se podrían trabajar rangos con letras de la siguiente manera

```
var resp='b' in 'a'..'e'  //almacena true en resp
println(resp)
```

***Instructor: Cristian David Henao Hoyos***