**Apuntes Android Kotlin**

**Variables vs. Objetos**

En **Kotlin** *variables vs. objetos* significa otra cosa en comparación a otros lenguajes de programación.

Una **variable** es un espacio en memoria que nosotros reservamos para almacenar un solo dato.

Un **objeto** es un espacio en memoria que es más complejo que una variable, se va a componer de otras variables y otros objetos, acciones, métodos y funcionalidades.

En **Kotlin** todo será un objeto, evitaremos el tipo de datos variables, simples o sencillos (*tipos primitivos*). No debemos declarar valores primitivos (*si podríamos pero no debemos*), pero son definidos cuando no los usamos como objetos.

Ejemplo de dato **primitivo**:

**var** i = 10

i = i \* 2

println(i)

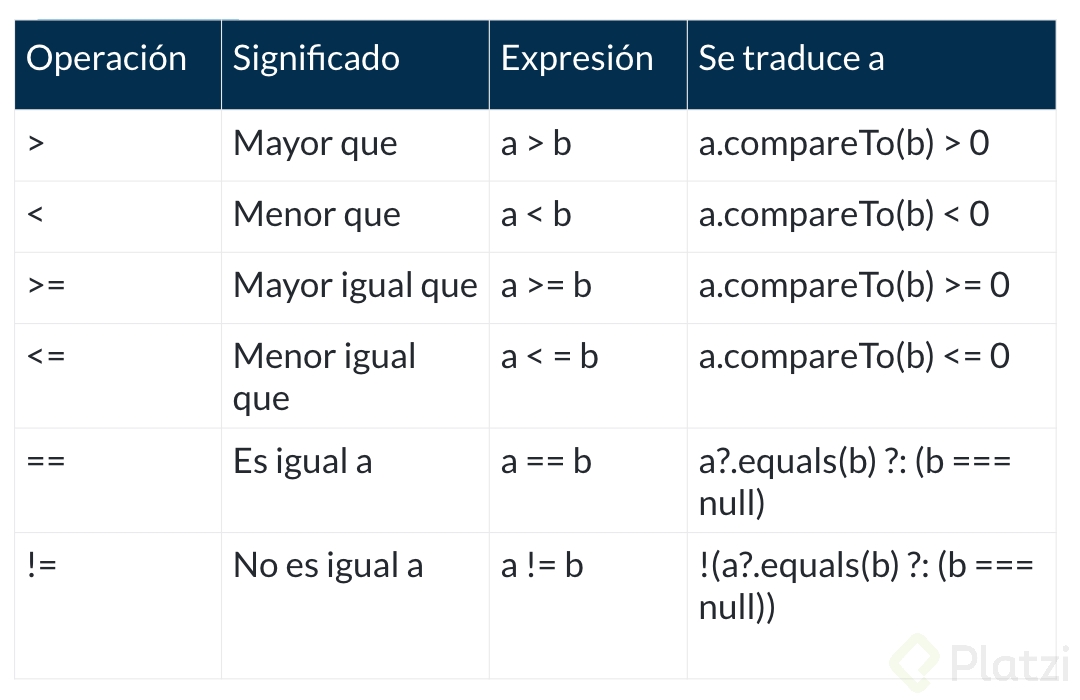
**Kotlin** utiliza *wrappers (clases envoltorio)* para los números, esto se llama *boxing*.

En los operadores de **Kotlin** debemos tratar todo como si fuera un objeto:

| **Expresión** | **Función** | **Se traduce a** |
| --- | --- | --- |
| a + b | plus | a.plus(b) |
| a - b | minus | a.minus(b) |
| a \* b | times | a.times(b) |
| a / b | div | a.div(b) |
| a % b | mod | a.mod(b) |
| a += b | a = a + b | a.plusAssign(b) |
| a -= b | a = a - b | a.minusAssign(b) |
| a \*= b | a = a \* b | a.timesAssign(b) |
| a /= b | a = a / b | a.divAssign(b) |
| a %= b | a = a % b | a.modAssign(b) |

| **Operador** | **Significado** | **Expresión** | **Se traduce a** |
| --- | --- | --- | --- |
| + | Suma unaria | +a | a.unaryPlus() |
| - | Resta unaria | -a | a.unaryMinus() |
| ! | Negación | !a | a.not() |
| ++ | Incremento | ++a | a.inc() |
| -- | Decremento | --a | a.dec() |

**Equidad**



Hagamos algunos ejemplos para entender cómo funcionan.

Los de asignación son sencillos de entender, basta con la explicación y el desglose que se proporciona en la tabla.

**Los operadores unarios** no harán más que seguir y tratar de aplicar la regla de los signos, en este caso puedes aplicar el operador o utilizar el método al cual Kotlin lo traduce. Algo así:

**var** **a** = -2

**var** **b** = **a**.unaryMinus()

**println**("b: $b")

Esto imprimirá:  
b: 2

Lo que hace es convertir el dato a positivo, pues siguiendo la regla de los signos (-) por (-) es (+).

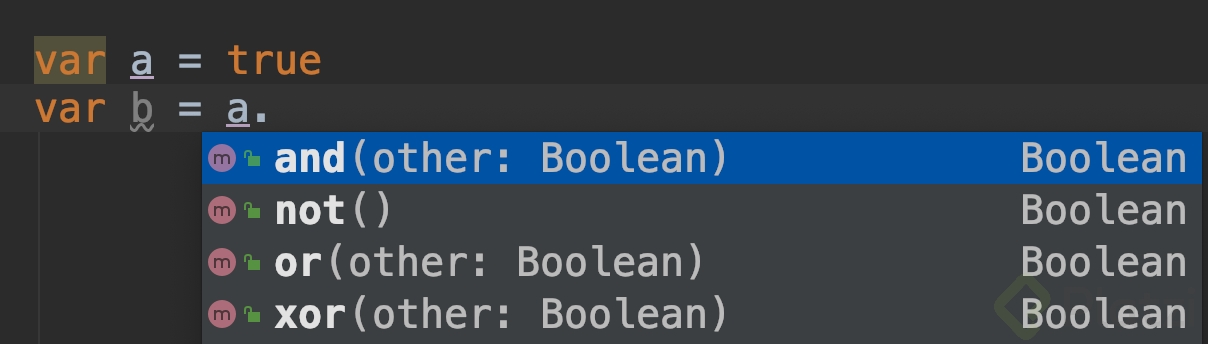
El operador de negación, funcionará con datos lógicos, este simplemente negará el dato establecido.

**var** **a** = true

**var** **b** = **a**.not()

**println**("b: $b")

El resultado será:  
b: false

Al utilizar datos lógicos, la función not() no será la única que podemos usar, tenemos otras opciones como se muestra en la figura:  


**El operador incremento y decremento.** Este se dedicará a incrementar o decrementar el valor de la variable en 1. Aquí ocurrirán dos momentos en que se puede incrementar o decrementar la variable, estos se llamarán: Pre-Incrementar a o Post-Incrementar a y Pre-Decrementar --a y Post-Decrementar a–.

Cuéntame en la sección de discusiones cuál es la diferencia entre usar Pre o Post.

Los operadores de equidad se enfocarán en compa1``rar si un dato es igual, o mayor, o mayor e igual que otro, etc. su resultado será un valor lógico. Estos pueden ser utilizados en los controladores de flujo como if, when, for, while etc.

val **a** = -12

val **b** = 12

**println**("a es mayor que b. ${a>b}")

**Tipos de variables: var, val y const**

Hay dos tipos de variables en **Kotlin**: *changeables (variables que pueden cambiar)* y *unchangeables (variables que no pueden cambiar)*

Las variables que **pueden cambiar** son declaradas con las palabra reservada var y las variables que **no pueden cambiar** son declaradas con la palabra reservada val.

Las const y las val son prácticamente lo mismo: valores que no se pueden cambiar. La diferencia entre estas dos es: el valor de const se determina en el tiempo de compilación y el valor de val se determina en tiempo de ejecución.

**Programación Funcional: Funciones Puras e Inmutabilidad**

Comencemos nuestro primer encuentro con la programación funcional. Primero entendamos qué es.

Como primera declaración debemos decir que la **programación funcional es un paradigma de programación**, esto significa, literalmente, un estilo de resolver problemas para plasmar la solución con código. Cuando escuchas la palabra **paradigma** probablemente lo primero que viene a tu mente es el Orientado a Objetos, y sí, precisamente ese es otro estilo de resolución de problemas. Programar de manera funcional significa que lo haremos de forma declarativa, es decir nos preocuparemos más por el **qué** que por el **cómo**.

Este estilo de programación tiene que ver con usar funciones todo el tiempo, podemos usarlas como parámetros de otras funciones e incluso asignarlas a variables. En este paradigma casi todo lo que habíamos aprendido sobre programación dejaremos de usarlo. Nos olvidaremos de usar variables como algo que puede cambiar, en su lugar creeremos en la **inmutabilidad**. Los bucles for, while, do while, etc. dejarán de existir pues ahora tendremos **recursividad**, además que, como ya dijimos, las funciones serán tan especiales que estas deberán ser **funciones puras** por definición.

## Funciones Puras

Una función pura, deberá cumplir con dos cosas específicamente.

**Primero**. Dados los mismos parámetros de entrada la función debe retornar siempre el mismo valor.

**Segundo**. La función no debe tener efectos colaterales, es decir no debe haber nada en el entorno que la altere. Como por ejemplo, variables globales que fue el ejemplo que vimos en la clase anterior.

## Inmutabilidad

Hablando de variables, hace un momento te comenté que estas las dejaremos de tratar como tal y fomentaremos el uso de **variables de tipo constante**, o variables inmutables.

La inmutabilidad es uno de los principios de la programación funcional donde nos promueve la ausencia de estado mutable o también conocido como **Stateless**, para entender esto mejor lo primero que debemos comprender es qué cosa se considera estado.

**Estado**: será cualquier dato que se pueda guardar y modificar posteriormente en memoria

* Una variable
* Un archivo
* Un socket

Para el caso de las variables en Kotlin preferirémos declararlas como ´val´ en vez de ´var´.  
Una buena práctica en nuestros programas es nunca tener variables de tipo ´var´ declaradas de manera **global** pues estas van en contra de la inmutabilidad, aunque sé que a veces es complicado prescindir de ellas en el resto del programa. Su mejor lugar es cuando están declaradas de manera local al método.

A lo largo del curso seguiremos aprendiendo pequeñas píldoras de conocimiento sobre la programación funcional.

**Strings**

**Raw String**: En Kotlin, además, tenemos cadenas sin procesar de tres comillas que pueden contener caracteres especiales sin la necesidad de escapar de ellas.

**.trimIndent()**: Detecta una sangría mínima común de todas las líneas de entrada, la elimina de cada línea y también elimina la primera y la última si están en blanco.

**.trimMargin()**: Recorta los caracteres de espacio en blanco iniciales seguidos de un *marginPrefix* de cada línea de una cadena de origen y elimina la primera y la última línea si están en blanco.

Puedes usar los siguientes para escapar caracteres especiales: \t, \b, \n, \r, \', \", \\ y \$