# Tema 0 . Programación Básica en Kotlin

## Índice


1. Índice
2. Introducción
3. Variables
4. Clases
5. Funciones

# Introducción

**Kotlin** es un lenguaje de programación fuertemente tipado desarrollado por **JetBrains** en 2010 y que está influenciado por lenguajes como **Scala , Groovy y C#**.
La mayoría de desarrollos con Kotlin, tienen como destino Android o la máquina virtual de java (JVM), y aunque también ofrece la posibilidad de desarrollos para la máquina virtual de JavaScript o código nativo, por ahora todavía son pocos los programadores que lo usan para estos destinos.
A partir de la actualización Kotlin 1.3.30, se incluyeron las mejoras para Kotlin/Native que permite compilar el código fuente de Kotlin en datos binarios independientes para diferentes sistemas operativos y arquitecturas de CPU, incluido IOS, Linux, Windows y Mac.

# Variables

Igual que en todos los lenguajes de programación, en Kotlin también tendremos el recurso de las variables para almacenar valores. En Kotlin nos podemos encontrar con variables **inmutables** y variables **mutables**. En el caso de las primeras, una vez se le asigna un valor a la variable no podrá ser modificado, es decir, se comporta como una constante (lo mismo que utilizar final en Java o const en C#). Mientras que con la mutables podremos modificar en cualquier momento el valor de la variable.
Para declarar una variable como mutable, la tendremos que preceder de la palabra clave **var** mientras que para las inmutables usaremos **val**.
El concepto de inmutabilidad es muy interesante. Al no poder cambiar los objetos estos son más robustos y eficientes. Siempre que podamos hemos de usar objetos inmutables.

```
//Variables mutables
var mutable:Int=5;
mutable+=7;
var numeroDecimales=3.14F;
numeroDecimales=3.12F;
//Variables inmutables
val inmutable:Char='C';
inmutable='A'  //Error compilación!! no se puede modificar una variable inmutable
```

> Como se puede ver en el ejemplo, cuando declaramos una variable, podemos indicar el tipo de esta o esperar a que el compilador lo infiera.

Para definir el tipo, tendremos que indicarlo con `:tipo` después del nombre:

**(var|val) nombreVariable [:tipo][=valor]**

Los tipos básicos de variables en Kotlin son:

| Tipo | Valor | Tamaño |
|---|---|---|
| Byte | 5.toByte() | 8 bits |
| Short | 5.toShort() | 16 bits |
| Int | 5 | 32 bits |
| Long | 5L | 64 bits |
| Float | 1.45F | 32 bits |
| Double | 1.45 | 64 bits |
| Boolean | true | |
| Char | 'H' | 16 bits |
| Unit | Unit | 0 bits |

> El tipo `Unit` corresponde a void en Java y C#

Cuando una **variable mutable** declarada en el cuerpo de una clase, no se quiere inicializar en el momento de la declaración, existe el concepto de `Inicialización tardía`, añadiendo el modificador **lateinit** delante de la declaración de la variable.

```
public class Ejemplo{
lateinit var cadena: String
fun miFuncion() {
  cadena = "Hola Mundo";
  println(cadena);}
}
```

# Clases

Todos los elementos en Kotlin son públicos por defecto, por lo que también lo serán las clases. Las clases nos sirven para referenciar objetos del mundo real, y mediante las propiedades podemos definir las distintas características que nos interese manipular con las cuales conste ese objeto; por ejemplo, para definir una clase Persona en Kotlin con algunas propiedades (sí, propiedades y no atributos) podemos hacer lo siguiente:

# Funciones

Si queremos crear una función en kotlin tendremos que precederla de la palabra reservada **fun**. Por tanto, una función constará de la palabra fun seguida por el nombre de la función y entre paréntesis los parámetros, siempre y cuando los tenga. Si la función retorna un valor se definirá al final de la signatura (esto último se puede omitir siempre que la función no devuelva nada).

**fun nombreFuncion([nombreVariable:Tipo, nombreVariable:Tipo, ... ]):TipoDevolucion**

```
fun sumaDatos(datoUno:Int, datoDos:Char )
{
   println("${datoUno + datoDos.toInt()}");
}
fun main() {
   var mutable:Int=5;
   mutable+=7;
   var numeroDecimales=3.14F;
  numeroDecimales=3.12F;
   val inmutable:Char='C';
   sumaDatos(mutable,inmutable);
}
```

Con valor de retorno:

```
fun mayor(numeroUno:Int, numeroDos:Int):Int
{
 return if(numeroUno>numeroDos)  numeroUno else numeroDos
}
fun main() { println(mayor(5, 7));}
```

Si la función es sencilla como el caso anterior. Kotlin pemite omitir las llaves.

```
fun mayor(numeroUno:Int, numeroDos:Int):Int= if(numeroUno>numeroDos)  numeroUno else num
```