Módulo 4:
¿A dónde vamos ? y Kotlin 01 (Variables, funciones)

<title> Objetivos del día </title>

- Bienvenido al Módulo 04.
- Kotlin
  - Variables.
  - Métodos.
  - Hola mundo en Kotlin.
  - Ejecutar un archivo Kotlin en Android Studio.
- Ejercicios Kotlin.

El camino hasta ahora...

¿Cuál es la competencia de este plan formativo?

“ Construir aplicaciones móviles Android que utilicen patrones de diseño de software escalable, persistencia de datos, se conecten a un servidor externo, y utilicen los elementos de interfaz de aplicaciones nativas Android, de acuerdo especificaciones dadas.”

Extraído directo del plan formativo sence.

¿Cuál era el objetivo del módulo 3?

“ Construir aplicaciones móviles Android utilizando lenguaje Java, patrones de diseño escalable y elementos de interfaz de aplicaciones nativas de acuerdo a las especificaciones entregadas”

Extraído directo del plan formativo sence.

¿Que podrías ser capaz de construir hasta ahora?

1. **Una app simple con fragmentos**
2. **Una app simple que solicite información al usuario y la guarde programáticamente en lo que dure la ejecución de la app.**
3. **Conocer el ciclo de vida de la Actividad y Fragmento.**
4. **Eventos sobre elementos de UI. (click listener)**
5. **Utilizar recursos, assets en una app.**
6. **Mostrar elementos (data estática) en un recyclerView.**
7. **Utilizar las Base de un patrón de arquitectura.**
8. **Realizar test unitarios.**
9. **Utilizar Control de versiones con Git.**

Módulo 4: Kotlin

¿Cuál es la competencia de este plan formativo?

“ Construir aplicaciones Android utilizando lenguaje Kotlin que puedan manejar persistencia de datos y conectarse a un servidor siguiendo especificaciones dadas”

Extraído directo del plan formativo sence.

¿Que vamos a aprender? Según Sence

- Construye una aplicación básica que se ejecute en el terminal acorde al lenguaje Kotlin.
- Construye una aplicación básica utilizando event listeners y view bindings acorde al lenguaje Kotlin.
- Construye un proyecto Android que maneja threads acorde al lenguaje Kotlin.
- Construye un proyecto Android que maneja persistencia de datos acorde a la librería ROOM.
- Construye un proyecto Android acorde al patrón MVP y LiveData.

¿Que vamos a aprender?

- Verifica el funcionamiento de una API REST acorde a una herramienta de requests HTTP (Postman)
- Construye un proyecto Android que consume un servicio REST y actualiza la interfaz de usuario acorde al lenguaje Kotlin y la librería Retrofit
- Construye un proyecto Android que consume un servicio REST que requiere autenticación acorde al lenguaje Kotlin, la librería Retrofit y auth token
- Verifica el funcionamiento de la integración entre la aplicación y el servicio REST utilizando lenguaje Kotlin.

¿Que vamos a aprender?

- Genera una firma .jks utilizando claves secretas para su posterior empaquetamiento
- Genera un apk de release utilizando un **crashlytics** para el seguimiento de caídas del aplicativo.

Introducción a Kotlin

Kotlin

- 2010: Creado por JetBrains
- 2017: Oficialmente soportado por Android.
- 2019: Anunciado por Google como lenguaje principal para Android.

Permite trabajar sobre:

- JVM
- JavaScript
- Kotlin Native

¿Por que Kotlin?

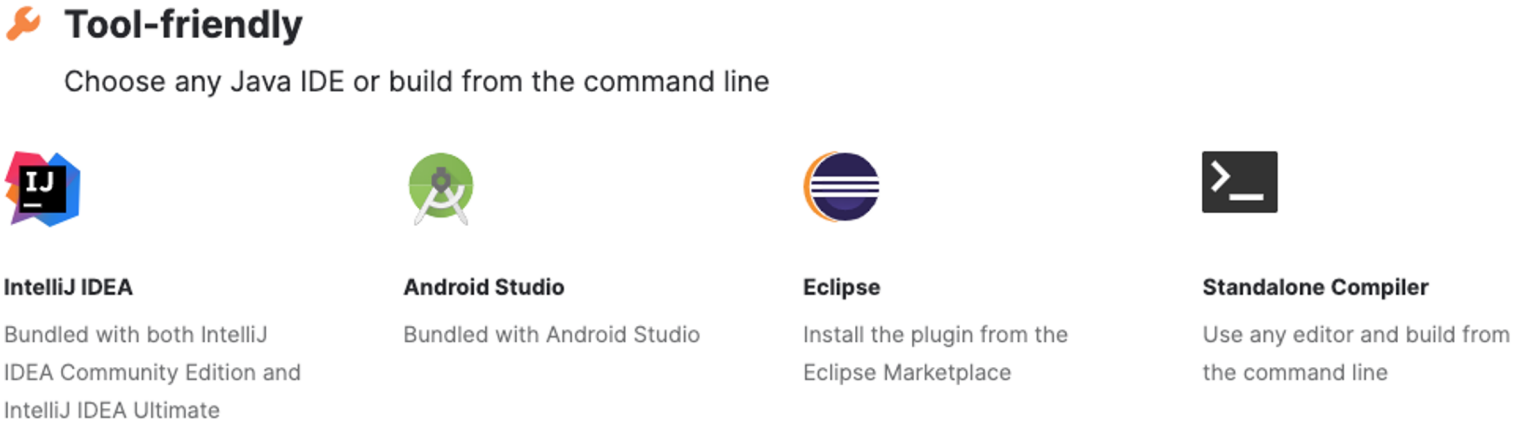
Conciso
Reduce drásticamente la cantidad de Boiler Plate

Seguro
Reduce muchos tipos de errores como los “Null Pointer exceptions”

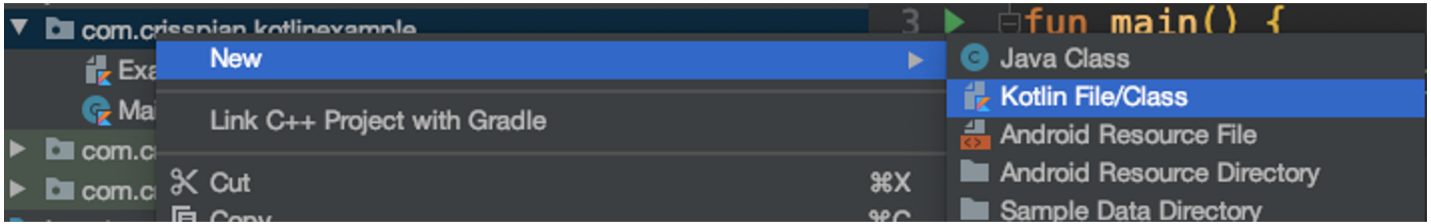
Interoperable
Puede operar con librerías existentes para la JVM, Android y el navegador.

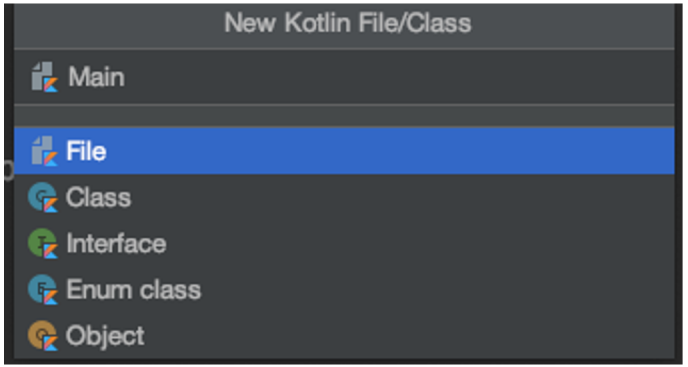
¿ Kotlin?

- Documentación oficial.



Ejercicio : ¡Hola Mundo en Kotlin! Android Studio.





1. Crear un proyecto en android studio.
2. Crear un nuevo Kotlin File/class

Ejercicio : ¡Hola Mundo en Kotlin! Android Studio.

1. El código kotlin está regularmente en packages. Si no especificamos alguno, se utilizará la ruta por defecto de los archivos.

1. El punto de entrada de una aplicación en Kotlin es la función **main**. Desde la versión 1.3 de Kotlin ya no se utiliza un array de string como tipo de retorno.

1. **println** es la forma estándar de mostrar un output, se importa de forma implícita.

El punto y coma final es Opcional.

```
package com.crisspian.kotlinexample // 1

fun main() {    // 2
   println("Hola mundo mundial...")  // 3
}
```

Variables

Variables en Kotlin

Variables en Kotlin

- ¿Qué es una variable?
- Declarar variables en Kotlin.
- Declarar variables por inferencia en Kotlin.
- Aprender formato de caracteres en Kotlin.

Declarar variables en Kotlin.

Para poder declarar una variable en Kotlin, necesitamos saber 3 cosas:

- Tipo de variable (“var” o “val”)
- Nombre de la variable
- Tipo de valor o Clase(Numérico, Cadena de Caracteres, Lógico, etc.)

** El segundo ejemplo está declarando y asignando el valor a una variable.

```
fun main() {
    val numero : Int
    //var variable : TipoVariable
}

.............

fun main() {
    var number : Int = 10 // **
}
```

Variables por inferencia

En Kotlin podemos omitir la declaración del tipo de valor, y esto lo hacemos inicializando la variable con un valor determinado.

```
fun main() {
    var number = 10
    val numberTwo = 20

    var name = "Rebeca"
    val lastName = "Ramirez"

    var decimalNumber = 1.0
    val bigDecimalNumber = 23.00f
}
```

Val o Var (mutable o inmutable)

Las variables inmutables se declaran como “**val**” y contienen un valor que “**No**” puede ser modificado.

Las variables mutables se declaran como “**var**” y estas variables “**Sí**” pueden ser modificadas.

Val o Var (mutable o inmutable)

Ejemplo:

```
fun main() {
    val fixNumber = 10
    var variableNumber = 10
}
```

String Template Kotlin

Con el signo “**$**” podemos concatenar valores.

```
fun main() {
    val x = "valor string"
    val template = "Concatenando **** $x **** valores"
    println(template)
}
```

```
Concatenando **** valor string **** valores
```

String Template Kotlin

También podemos llamar a una función del lenguaje o de una variable utilizando el signo “**$**” seguido del uso de llaves “**{ }**”

[Documentación text]

```
fun main() {
    val x = "valor string"
    val template = "Tiene ${x.count()} caracteres"
    println(template)
}
```

```
Tiene 12 caracteres
```

Ejercicio: Probando Variables

Crear una función main.

- Dentro de la función main declarar una
variables mutable y una variable inmutable
de forma inteligente.
- Modifica el valor de la variable mutable.
- Imprime el valor por pantalla de las dos
variables.

```
fun main() {
    var a = 10
    val e = "Goku"
    a = 20
    println("Variable mutable: $a")
    println("Variable inmutable: $e")
}
```

```
Variable mutable: 20
Variable inmutable: Goku

Process finished with exit code 0
```

Tipos de datos numéricos enteros en Kotlin

Tipo	Tamaño	Rango de valor
Byte	8 bits	-128 a 127
Short	16 bits	-32768 a 32767
Int	32 bits	-2147483648 a 2147483647
Long	64 bits	-9223372036854775808 a 9223372036854775807

Tipos de datos numéricos enteros

En cada clase Byte, Short, Int y Long existe una constante llamada MAX_VALUE y MIN_VALUE.

```
fun main() {
    val byteMaximo = Byte.MAX_VALUE
    val byteMinimo = Byte.MIN_VALUE
    val shortMinimo = Short.MIN_VALUE
    val shortMaximo = Short.MAX_VALUE
    val intMinimo = Int.MIN_VALUE
    val intMaximo = Int.MAX_VALUE
    val longMinimo = Long.MIN_VALUE
    val longMaximo = Long.MIN_VALUE
    println(shortMaximo)
}
```

Tipos de datos coma flotante

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

Tipos de datos coma flotante

En Kotlin existen 2 tipos de variables básicas para trabajar con valores de coma flotante estos son Float y Double.

```
fun main() {
    val floatNumber = 19.5f
    var doubleNumber : Double = 20.5
}
```

Tipos de datos Booleanos

Las variables booleanas almacenan sólo dos valores, “verdadero” o “falso”

|| – lazy disjunction “O”
&& – lazy conjunction “Y”
! - negation

```
fun main() {
    val x = true
    val y : Boolean
}
```

Tipos de datos cadena de caracteres o char

En Kotlin existen dos tipos básicos para trabajar Textos , cadenas de caracteres String y Char.

```
fun main() {
    val x = 'a'
    val y = "Cadena de caracteres"
    var vocal : Char
    val nombre : String
    vocal = 'u'
    nombre = "Chizuru Kagura"
}
```

Ejercicio: Probando Variables Básicas

Crear una función main.

- Dentro de la función main declarar una
variable de tipo Int, String, Char y Float.
- Imprime el valor por pantalla de todas las variables.

```
fun main() {
    val entera = 10
    val cadena = "Goku"
    val vocal = 'U'
    val flotante = 10F
    println("Variable Int: $entera")
    println("Variable String: $cadena")
    println("Variable Char: $vocal")
    println("Variable Float: $flotante")
}
```

```
Variable Int: 10
Variable String: Goku
Variable Char: U
Variable Float: 10.0

Process finished with exit code 0
```

Ejercicio: Probando Variables numéricas

- Crear una función main.
- Dentro de la función main declarar tres variables numéricas con los valores 10, 33, 66.
- Declara una variable resultado que guarde la suma de las 3 variables.
- Imprime la variable resultado.
- Modificar la variable con el valor 10, por el valor 55.
- Imprimir nuevamente variable resultado.
- Declarar una variable promedio, que calcule el promedio de las 3 variables.
- Imprimir promedio.

```
fun main() {
    var a = 10
    val b = 33
    val c = 66
    var resultado = a + b + c
    println("$a + $b + $c = $resultado")
    a = 55
    resultado = a + b + c
    println("$a + $b + $c = $resultado")
    val promedio = resultado/3
    print("Promedio : $promedio")
}
```

```
10 + 33 + 66 = 109
55 + 33 + 66 = 154
Promedio : 51
Process finished with exit code 0
```

Funciones

Funciones en Kotlin

En Kotlin, solo existen los métodos con tipo de retorno y a estos métodos se les llama funciones.

En Kotlin a diferencia de Java se dice que las funciones son de primer orden por tanto puede generarlos fuera de una clase y también dentro de otra función.

Una función debe comenzar con la palabra reservada fun, luego un nombre, un paréntesis con parámetros y luego el tipo de retorno seguido de un cuerpo {}

```
fun getNombreCompleto(): String{
    return "Jenniffer Cornell"
}
```

¿Funciones Void?

En Kotlin no se utiliza la palabra reservada “**void**”, sólo existe la clase “Void”, pero si le indicamos a una función que retornará un objeto tipo “Void” nos llevaremos una sorpresa.

```
fun imprimirNombreCompleto(): Unit{
    println("Jenniffer Cornell")
}
```

Funciones con Parámetros

La forma de declarar parámetros en una función Kotlin es la siguiente:

```
fun imprimirNombreCompleto(nombres: String, apellidos: String, edad: Int){
    println("Nombres: $nombres")
    println("Apellidos: $apellidos")
    print("Edad: $edad")
}
```

Más Funciones.

1. Una función simple que toma un parámetro del tipo string y retorna Unit.
2. Una función que retorna un segundo parámetro opcional. El tipo de retorno fue omitido, lo que quiere decir que es Unit.
3. Una función que retorna un Entero.
4. Una función simple que retorna un entero (está infiriendo el tipo de retorno)
5. Llamado a la función con el argumento (“hello)
6. LLamando a una función pasando dos parámetros.
7. La misma de antes pero omitiendo un parámetro
8. La misma función usando el argumentos nombrados, y cambiando el orden de los argumentos.
9. Imprime el resultado de una llamada al la función.

```
fun printMessage(message: String): Unit {                    // 1
    println(message)
}

fun printMessageWithPrefix(message: String, prefix: String =
"Info") {  // 2
    println("[$prefix] $message")
}

fun sum(x: Int, y: Int): Int {                               // 3
    return x + y
}

fun multiply(x: Int, y: Int) = x * y                         // 4

fun main() {
    printMessage("Hello")                                    // 5
    printMessageWithPrefix("Hello", "Log")                   // 6
    printMessageWithPrefix("Hello")                          // 7
    printMessageWithPrefix(prefix = "Log", message = "Hello")
// 8
    println(sum(1, 2))                                       // 9
}
```

Funciones con varArg

1. Una función con la palabra reservada vararg puede recibir varios argumentos
2. Esto permite imprimir varios tipos de argumentos.
3. Gracias a los parámetros nombrados, se pueden añadir otros parámetros del mismo tipo al final de varargs.
4. Usando los parámetros nombrados, puedes añadir un prefix separado de los argumentos.

```
fun printAll(vararg messages: String) {                    // 1
    for (m in messages) println(m)
}


fun printAllWithPrefix(vararg messages: String, prefix:
String) {  // 3
    for (m in messages) println(prefix + m)
}




fun main() {
    printAll("Hello", "Hallo", "Salut", "Hola", "你好")
// 2

    printAllWithPrefix(
        "Hello", "Hallo", "Salut", "Hola", "你好",
        prefix = "Greeting: "                              // 4
    )
}
```

Funciones de extensión

En este ejemplo podemos ver cómo declaramos una función llamada String.saludo cuando al nombre de nuestra función le anteponemos el nombre de una clase (en este caso la clase String) significa que cualquier objeto de la clase String puede utilizar nuestra función saludo y despedida. Utilizarla en Main.

```
fun String.saludo() {
    println("saludo")
}

fun String.despedida() {
    println("chao")
}
```

```
fun main(){
    val variable = "Hola" //declaramos una variable String
    variable.saludo() //Utilizamos nuestra función de extención "saludo"
    variable.despedida() //Utilizamos nuestra función de extención "despedida"
}
```

Ejercicio: Probando Funciones

- Crear una función llamada getNombre que devuelva un String con tu nombre.
- Crea una función llamada “imprimirNombreCompleto” que reciba los siguientes parámetros: “nombres”, “apellidos”, edad.
- En una función main() llame a estas dos funciones.