UT1.- Introducción a Kotlin para Android

Introducción

- Lenguaje de programación de propósito general.
- Lenguaje de tipo estático para Java Virtual Machine (JVM) y JavaScript.
- Sintaxis amigable (híbrida entre Java, C# y Javascript).
- Extensión ficheros *.kt* .
- Las sentencias NO acaban en “;”
- Máxima ▫ mayor productividad, mejorando la experiencia de codificación de una manera práctica y eficaz.
- Por la experiencia de los alumnos que han hecho prácticas en equipos o departamentos de “Movilidad”, si se hace una aplicación nueva en Android Nativo, se hace con Kotlin (en java queda el mantenimiento de aplicaciones que se hicieron en Java).

Comentarios

- Al igual que la mayoría de los lenguajes modernos, Kotlin admite comentarios de una sola línea (o final de línea) y de varias líneas (bloque).

```
// This is an end-of-line comment
/* This is a block comment
 * on
 * multiple
 * lines. */
```

Constantes (val) y Variables (var)

```java
// Java
int a = 1;
String b = "xyz";


// Kotlin
val a = 1        // Inferred type is Int
val b = "xyz"    // Inferred type is String
```

```
val name:String
name = "DA2D1E"
```

```
var name: String
name = "DA2D1E"
```

```
var name = "DA2D1E"
```

- Al declarar una constante, si le asignas directamente el valor NO es necesario indicar el tipo de dato, lo infiere del valor asignado.
- Si NO asignas el valor directamente, entonces es necesario indicar el tipo de la constante.

Constantes (val) y Variables (var)

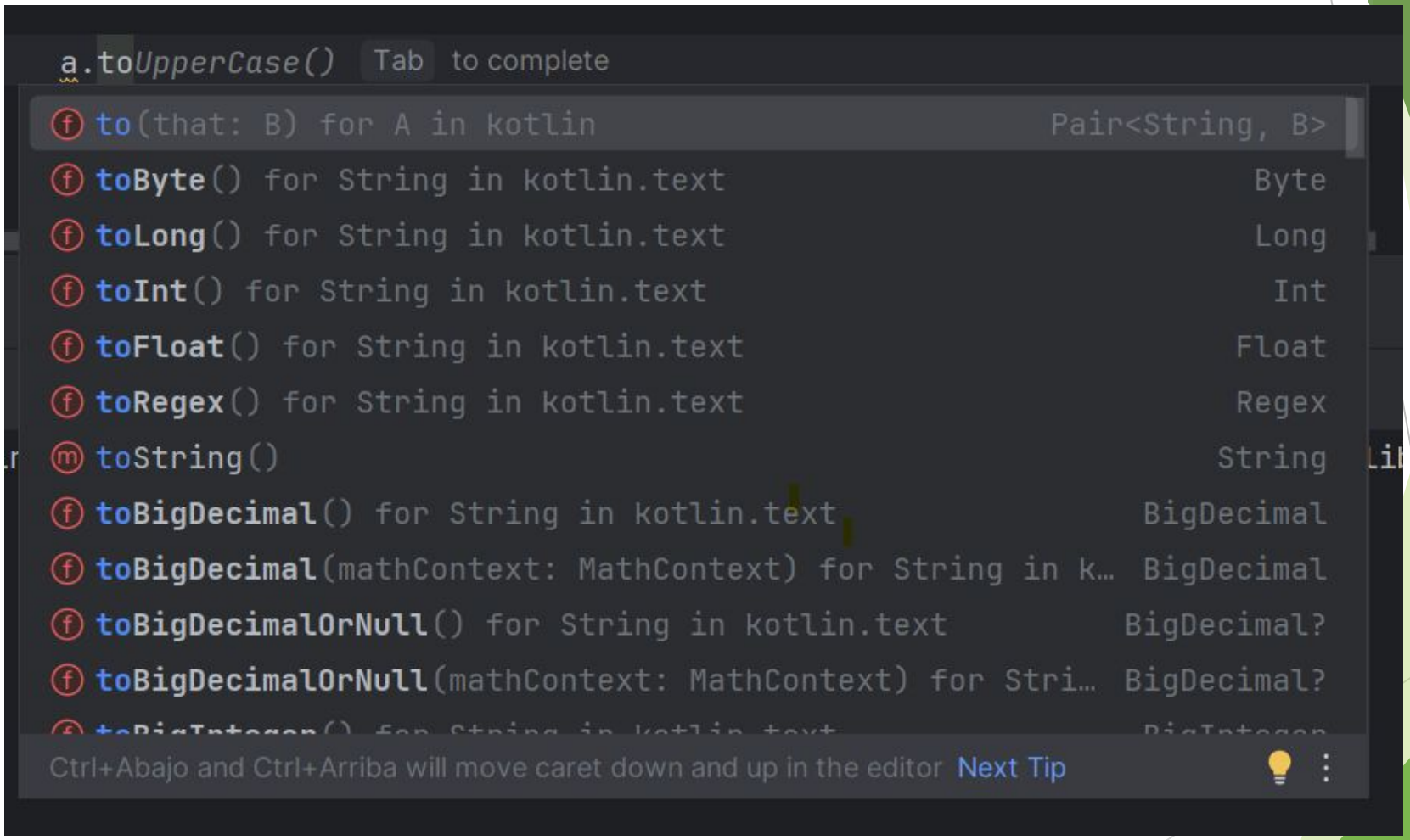
```
// Assign-once (read-only) variable
val x = 1
x = 2       // Compile-time error


// Mutable variable
var y = 5
y = 10      // OK


// Always start with immutable "val"
// Change to "var" only when necessary
```

There are four visibility modifiers in Kotlin: `private`, `protected`, `internal`, and `public`. The default visibility is `public`.

Constantes (val) y Variables (var)

Tipo de datos de Kotlin	Qué tipo de datos puede contener	Ejemplos de valores literales
`String`	Texto	`"Add contact"` `"Search"` `"Sign in"`
`Int`	Número entero	`32` `1293490` `-59281`
`Double`	Número decimal	`2.0` `501.0292` `-31723.99999`
`Float`	Número decimal (que es menos preciso que un `Double`). Tiene un `f` o `F` al final del número.	`5.0f` `-1630.209f` `1.2940278F`
`Boolean`	`true` o `false`. Usa este tipo de datos cuando solo haya dos valores posibles. Ten en cuenta que `true` y `false` son palabras clave en Kotlin.	`true` `false`

Variables (var): conversiones a otros tipos



Operadores de signo

Operador	Expresión	Función equivalente
+	*+a*	a.unaryPlus()
-	*-a*	a.unaryMinus()

```
var a = -3

println("a = $a")
println("a = ${-a}")
```

```
C:\Users\jhorn\.j
a = -3
a = 3
```

Operadores aritméticos

Operador	Operación	Expresión	Función Equivalente
+	Suma	a+b	a.plus(b)
-	Resta	a-b	a.minus(b)
*	Multiplicación	a*b	a.times(b)
/	División	a/b	a.div(b)
%	Resto	a%b	a.rem(b)

Operadores aritméticos

```kotlin
fun main() {
    val a = -10
    val b = 30

    println("($a + $b)= ${a + b}")
    println("($a - $b)= ${a - b}")
    println("($a * $b)= ${a * b}")
    println("($a / $b)= ${a / b}")
    println("($a % $b)= ${a % b}")
}
```

```
(-10 + 30)= 20
(-10 - 30)= -40
(-10 * 30)= -300
(-10 / 30)= 0
(-10 % 30)= -10
```

```kotlin
fun main() {
    val a = -10
    val b = 30

    println("($a / $b)= ${a.toDouble()/b}")
}
```

```
(-10 / 30)= -0.3333333333333333
```

```kotlin
fun main() {
    val a = -10
    val b = 30.0

    println("($a / $b)= ${a / b}")
}
```

```
(-10 / 30.0)= -0.3333333333333333
```

Operadores de incremento/decremento

Operador	Expresión prefijo	Expresión sufijo
++	++a	a++
--	--a	a--

Operadores de incremento/decremento

```
fun main() {
    var a = 2

    println("De $a a ${++a}")
    println("De $a a ${a--}")
    println("Valor final > $a")
}
```

```
De 2 a 3
De 3 a 3
Valor final > 2
```

Operadores de asignación compuesta

Operador	Expresión simplificada	Expresión Completa	Función Equivalente
+=	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.remAssign(b)

Operadores de asignación compuesta


```kotlin
fun main() {
    var a = -10
    val b = 30

    println("Valor de a = ${a}")
    a += b // a = a + b
    println("Valor de a tras incremento de b = ${a}")
}
```

```
Valor de a = -10
Valor de a tras incremento de b = 20
```

Operadores relacionales

Operador	Enunciado	Expresión	Función Equivalente
==	*a* es igual a *b*	a==b	a?.equals(b) ?: (b === null)
!=	*a* es diferente de *b*	a!=b	!(a?.equals(b) ?: (b === null))
<	*a* es menor que *b*	a<b	a.compareTo(b)<0
>	*a* es mayor que *b*	a>b	a.compareTo(b)>0
<=	*a* es menor ó igual que *b*	a<=b	a.compareTo(b)<=0
>=	*a* es mayor o igual que *b*	a>=b	a.compareTo(b)>=0

Operadores relacionales

```kotlin
fun main() {
    val a = 17
    val b = 20

    println("$a es igual a $b: ${a == b}")
    println("$a es diferente a $b: ${a != b}")
    println("$a es menor que $b: ${a < b}")
    println("$a es mayor que $b: ${a > b}")
    println("$a es menor o igual que $b: ${a <= b}")
    println("$a es mayor o igual que $b: ${a >= b}")
}
```

```
17 es igual a 20: false
17 es diferente a 20: true
17 es menor que 20: true
17 es mayor que 20: false
17 es menor o igual que 20: true
17 es mayor o igual que 20: false
```

Operadores lógicos

Operador	Descripción	Expresión
&&	Conjunción (and): el resultado es true si *a* y *b* son true	a&&b
||	Disyunción (or): el resultado es true si *a* o *b* son true	a||b
!	Negación (not): el resultado es false si *a* es true, o viceversa	!a

Operadores lógicos

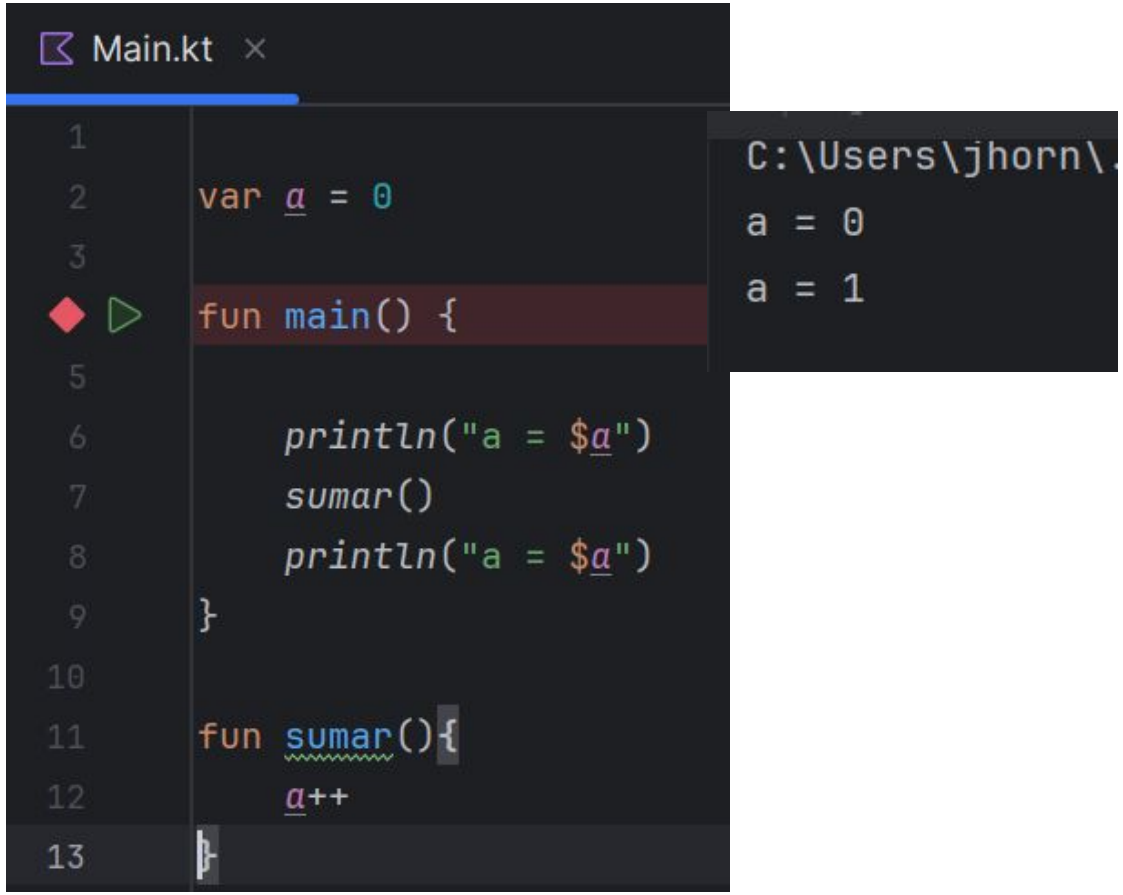
```kotlin
fun main() {
    val input = 5
    var res: Boolean

    val greaterThanZero = input > 0
    val isEven = input % 2 == 0

    res = greaterThanZero && isEven
    println("Es mayor que cero y par:$res")

    res = greaterThanZero || isEven
    println("Es mayor que cero o par:$res")

    res = greaterThanZero && !isEven
    println("Es mayor que cero e impar:$res")
}
```

```
Es mayor que cero y par:false
Es mayor que cero o par:true
Es mayor que cero e impar:true
```

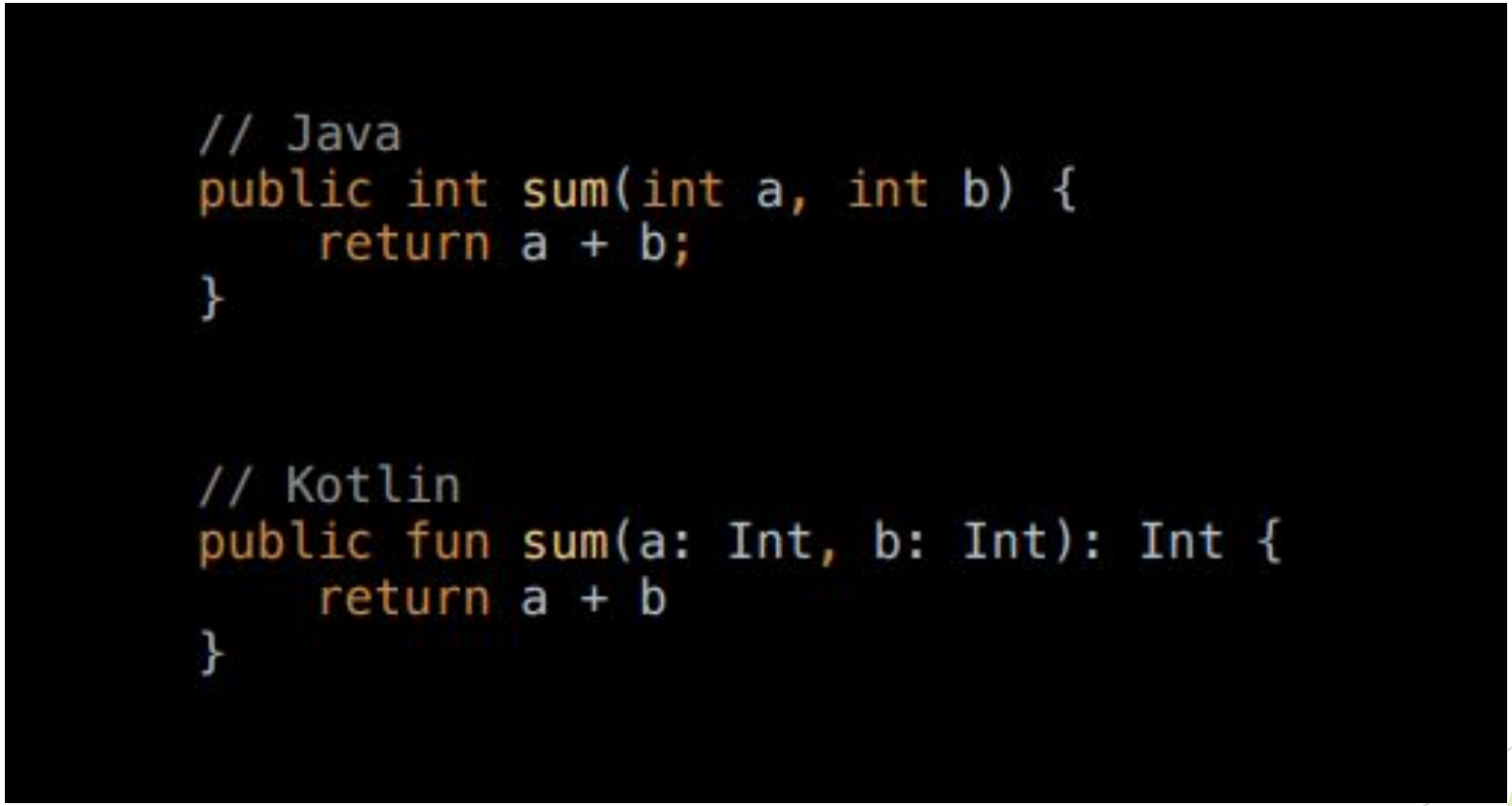
Funciones


fun main() {
println("Hello World")
}
Hello World
fun main(args: Array<String>) {
println(args.contentToString())
}
[]

Funciones


Main.kt
1
2 var a = 0
3
fun main() {
5
6 println("a = $a")
7 sumar()
8 println("a = $a")
9 }
10
11 fun sumar(){
12 a++
13 }
C:\Users\jhorn\
a = 0
a = 1

Funciones



```java
// Java
public int sum(int a, int b) {
    return a + b;
}
```

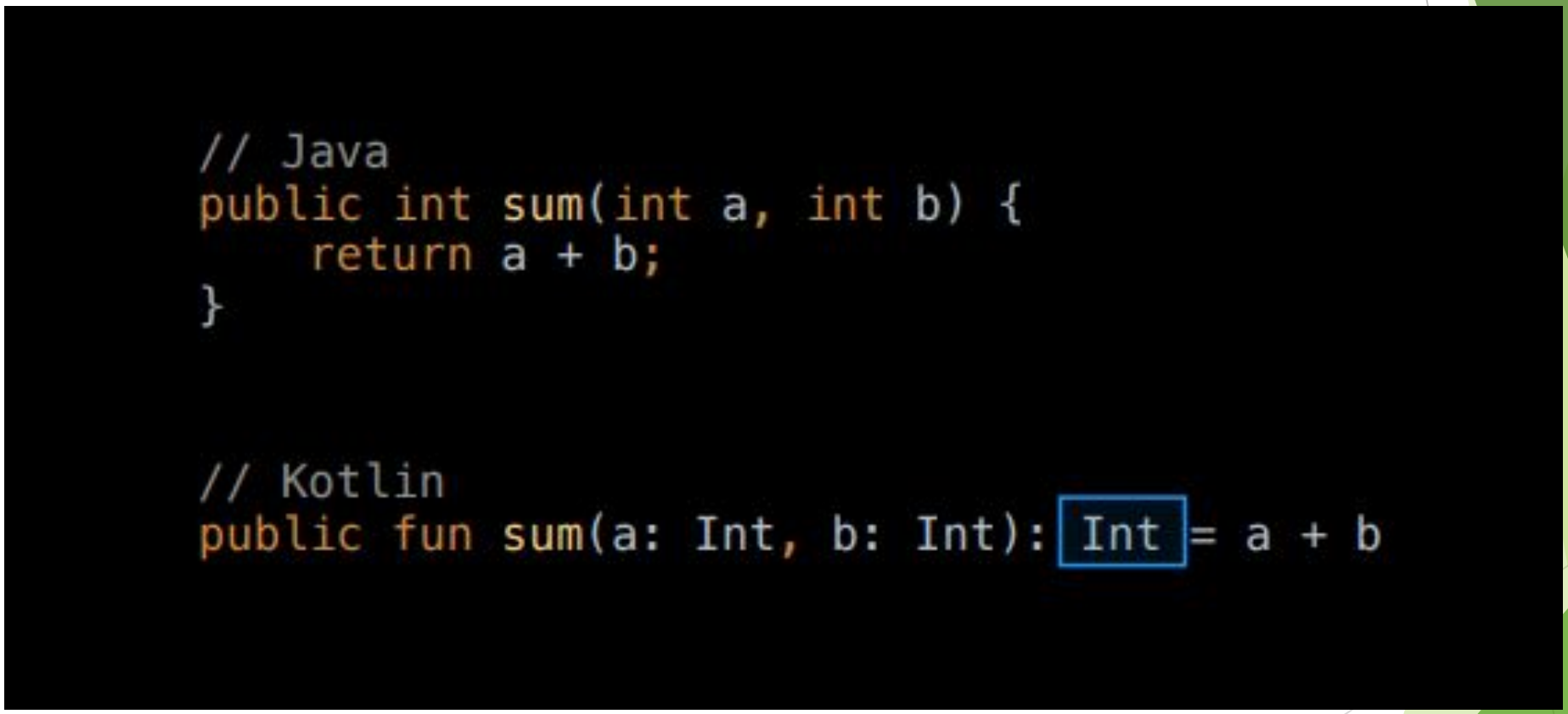
```kotlin
// Kotlin
public fun sum(a: Int, b: Int): Int {
    return a + b
}
```

Funciones

```kotlin
fun main() {
    var a = 0
    println("a = $a")
    a = sumar(a)
    println("a = $a")
}

fun sumar(x: Int ):Int {
    return x+1
}
```

```
C:\Users\jhorn\.
a = 0
a = 1
```

Funciones

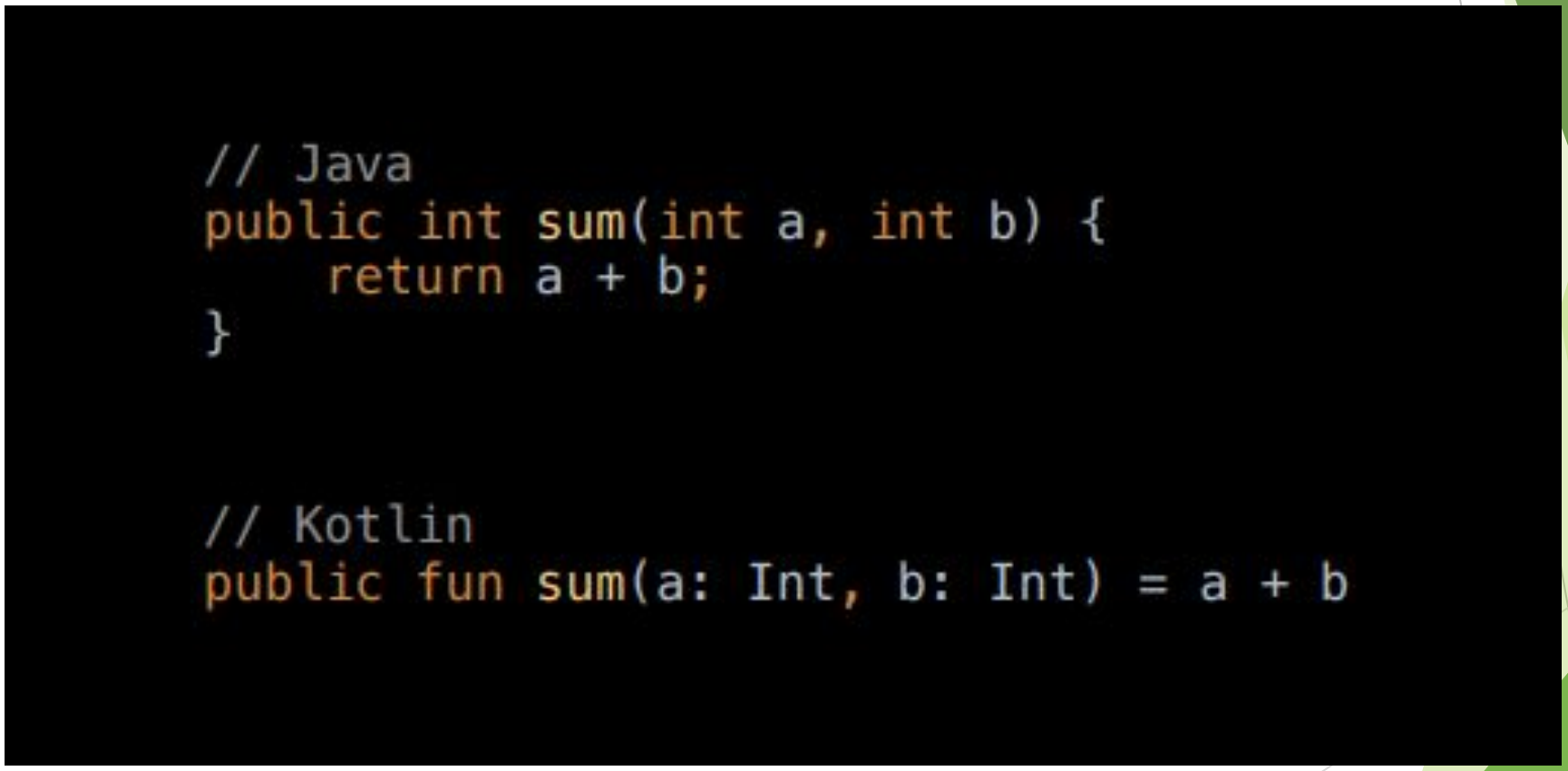

// Java
public int sum(int a, int b) {
return a + b;
}
// Kotlin
public fun sum(a: Int, b: Int): Int = a + b

Funciones

```kotlin
fun main() {
    var a = 0
    println("a = $a")
    a = sumar(a)
    println("a = $a")
}

fun sumar(x: Int ): Int = x+1
```

```
C:\Users\jhorn\.
a = 0
a = 1
```

Funciones


// Java
public int sum(int a, int b) {
return a + b;
}
// Kotlin
public fun sum(a: Int, b: Int) = a + b

Funciones

```kotlin
fun main() {
    var a = 0
    println("a = $a")
    a = sumar(a)
    println("a = $a")
}

fun sumar(x: Int ) = x+1
```

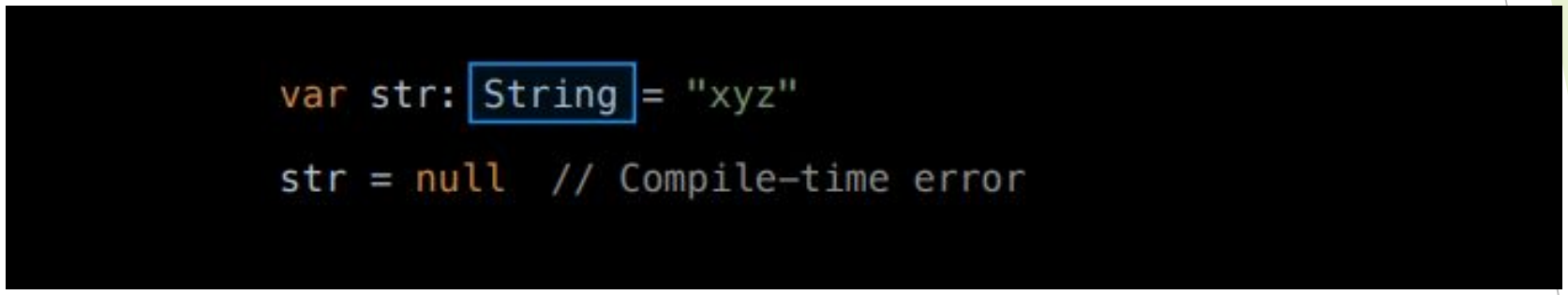
```
C:\Users\jhorn\.
a = 0
a = 1
```

Funciones y argumentos

```
Hello
[Log] Hello
[Info] Hello
[Log] Hello
3
8
```

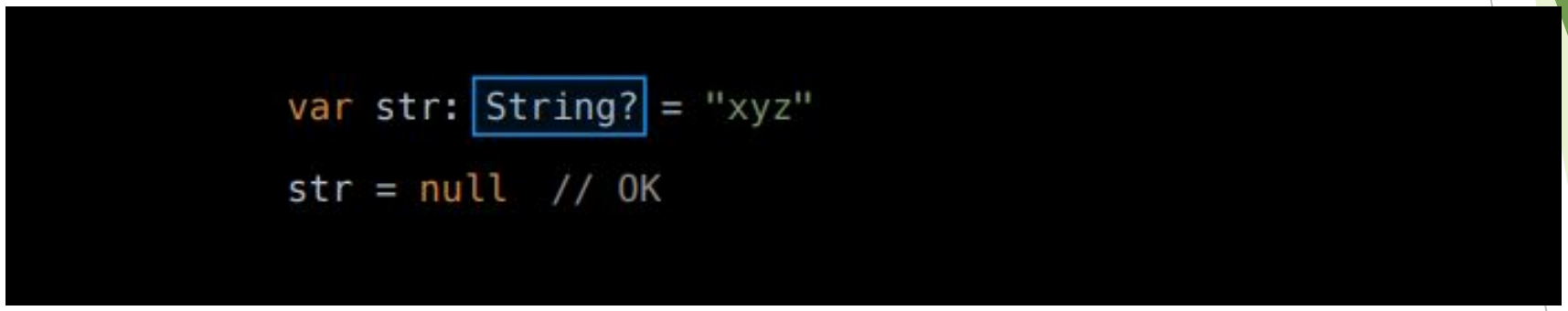
```kotlin
fun printMessage(message: String): Unit {
    println(message)
}

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

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

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

fun main() {
    printMessage("Hello")
    printMessageWithPrefix("Hello", "Log")
    printMessageWithPrefix("Hello")
    printMessageWithPrefix(prefix = "Log", message = "Hello")
    println(sum(1, 2))
    println(multiply(2, 4))
}
```

Null safety



- En un esfuerzo por deshacerse del *NullPointerException*, los tipos de variable en Kotlin no permiten la asignación de **null**.

Null safety



- Si se necesita una variable que pueda ser nula, hay que declararlo de manera explícita agregando **?** al final de su tipo.

Null safety

```
fun getLength(str: String): Int? {
    return str.length   // OK
}
```

```
fun getLength(str: String?): Int? {
    return str.length   // Compile-time error
}
```

- Si se indica que un parámetro o variable puede ser nulo, al intentar acceder a alguna de sus propiedades obtenemos un error indicándolo.

Null safety

```
fun getLength(str: String?): Int? {
    if (str != null) {
        return str.length    // <-- Smart cast
    }
    return 0
}
```

```
fun getLength(str: String?): Int? {
    return str?.length
}
```

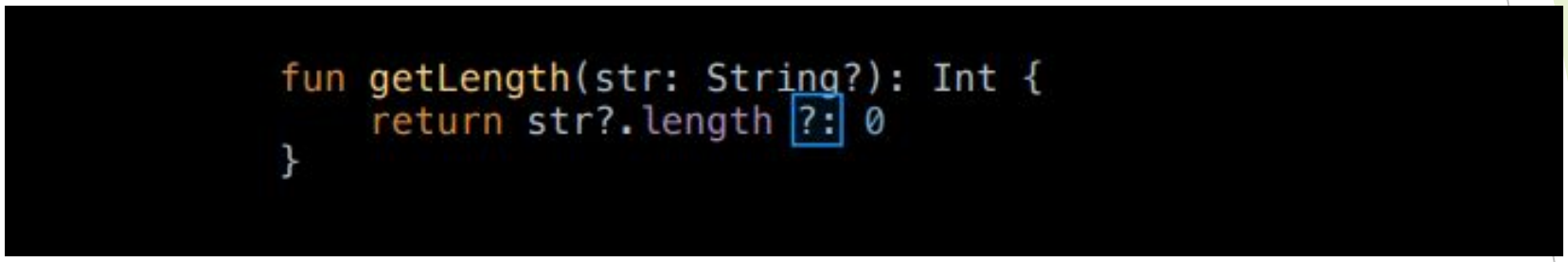
- Si existe la posibilidad de que la variable sea nula, es necesario utilizar el operador “llamada segura” **?.**

Null safety

```
fun main() {
    var x: String?
    x= null
    var a: Int?

    a = getLength(x)
    println("a = $a")
    x = "mi nombre"
    a = getLength(x)
    println("a = $a")
}

fun getLength(cadena:String?): Int? {
    return cadena?.length
}
```

```
C:\Users\jhorn\.jc
a = null
a = 9
```

Null safety



- Si la expresión a la izquierda de **?:** no es nula, es utilizada; de lo contrario, se utiliza la expresión de la derecha.

Null safety

```
fun main() {
    var x: String?
    x= null
    var a: Int?

    a = getLength(x)
    println("a = $a")
    x = "mi nombre"
    a = getLength(x)
    println("a = $a")
}

fun getLength(cadena:String?): Int? {
    return cadena?.length?: 9999
}
```

```
C:\Users\jhorn\.
a = 9999
a = 9
```

Null safety

```java
// Java
public ZipCode getZipCode(User user) {
    if (user != null) {
        if (user.getAddress() != null) {
            return user.getAddress().getZipCode();
        }
    }
    return null;
}
```

```kotlin
// Kotlin
fun getZipCode(user: User?): ZipCode? {
    return user?.address?.zipCode
}
```

Null safety

Kotlin

```java
// Java
public ZipCode getZipCode(User user) {
    if (user != null) {
        if (user.getAddress() != null) {
            return user.getAddress().getZipCode();
        }
    }
    return null;
}
```

```kotlin
// Kotlin
fun getZipCode(user: User?) = user?.address?.zipCode
```

Scanner


```
val sc = Scanner(System.`in`)
```

- El uso de Scanner es igual a Java. De hecho se importa la clase de java y los métodos (nextLine o nextInt) son los mismos.

Arrays

- Los arrays, al igual que en Java, deben tener un tamaño conocido. Se pueden inicializar con contenido:

```
val rockPlanets = arrayOf<String>("Mercury", "Venus", "Earth", "Mars")
```

- O se pueden inicializar vacíos y darles contenido en el propio programa
- El acceso al contenido del array se hace, como en Java, por el índice.

```
val planets = arrayOfNulls<String>(8)
planets[0] = "Mercurio"
planets[1] = "Venus"
planets[2] = "Tierra"
planets[3] = "Júpiter"
planets[4] = "Saturno"
planets[5] = "Urano"
planets[6] = "Neptuno"
planets[7] = "Plutón"
```

Más adelante veremos cómo se trabaja con listas.

String

```kotlin
fun main() {
    var a = 1
    // simple name in template:
    val s1 = "a is $a"

    a = 2
    // arbitrary expression in template:
    val s2 = "${s1.replace("is", "was")}, but now is $a"
    println(s2)
}
```

```
a was 1, but now is 2
```

String

```
override fun toString(): String {
    return "Song{id=" +
            id +
            ", title='" +
            title +
            "', author='" +
            author +
            "'}"
}
```

```
override fun toString(): String {
    return "Song{id=$id, title='$title', author='$author'}"
}
```

```
override fun toString() = "Song{id=$id, title='$title', author='$author'}"
```

Sentencias condicionales (if-else)


```kotlin
fun maxOf(a: Int, b: Int): Int {
    if (a > b) {
        return a
    } else {
        return b
    }
}

fun main() {
    println("max of 0 and 42 is ${maxOf(0, 42)}")
}
```

```
max of 0 and 42 is 42
```

Sentencia condicional (when)

```kotlin
fun describe(obj: Any): String =
    when (obj) {
        1          -> "One"
        "Hello"    -> "Greeting"
        is Long    -> "Long"
        !is String -> "Not a string"
        else       -> "Unknown"
    }

fun main() {
    println(describe(1))
    println(describe("Hello"))
    println(describe(1000L))
    println(describe(2))
    println(describe("other"))
}
```

```
One
Greeting
Long
Not a string
Unknown
```

Bucles (for)

```
fun main() {
    for(i in 0..8) {
        print(i)
    }
    println(" ")

    for(i in 0..8 step 2) {
        print(i)
    }
    println(" ")

    for(i in 0 until 8) {
        print(i)
    }
    println(" ")

    for (i in 8 downTo 0) {
        print(i)
    }
    println(" ")
}
```

```
012345678
02468
01234567
876543210
```

Bucles (for)

```
fun main() {
    val items = listOf("apple", "banana", "kiwifruit")
    for (item in items) {
        println(item)
    }
}
```

```
apple
banana
kiwifruit
```

Bucles (for)

```kotlin
fun main() {
    val items = listOf("apple", "banana", "kiwifruit")
    for (index in items.indices) {
        println("item at $index is ${items[index]}")
    }
}
```

```
item at 0 is apple
item at 1 is banana
item at 2 is kiwifruit
```

Bucles (while)

```kotlin
fun main() {
    val items = listOf("apple", "banana", "kiwifruit")
    var index = 0
    while (index < items.size) {
        println("item at $index is ${items[index]}")
        index++
    }
}
```

```
item at 0 is apple
item at 1 is banana
item at 2 is kiwifruit
```

Bucles (do while)


```
do {
    val y = retrieveData()
} while (y != null) // y is visible here!
```

Excepciones

```
try {
    // Code that may throw an exception
}
catch (e: YourException) {
    // Exception handler
}
finally {
    // Code that is always executed
}
```

Rangos (*in*)


```
fun main() {
    val x = 10
    val y = 9
    if (x in 1..y+1) {
        println("fits in range")
    }
}
```

```
fits in range
```

```
fun main() {
    val list = listOf("a", "b", "c")

    if (-1 !in 0..list.lastIndex) {
        println("-1 is out of range")
    }
    if (list.size !in list.indices) {
        println("list size is out of valid list indices range, too")
    }
}
```

```
-1 is out of range
list size is out of valid list indices range, too
```