



# Programación multimedia y dispositivos móviles

UT-3.1 - Funciones lambda o flecha en Kotlin



## Sintaxis De Lambdas En Kotlin

Una **función lambda** es un literal de función que puede ser usado como expresión. Esto quiere decir, una función que no está ligada a un identificador y que puedes usar como valor.

Por ejemplo, si tenemos la función `f(s) = s + 2`, en Kotlin podemos expresarla como una declaración de función separada, así:

```
fun sumarDos(s: Int) = s + 2
```



Al ser reescrita como lambda, tendrías lo siguiente:

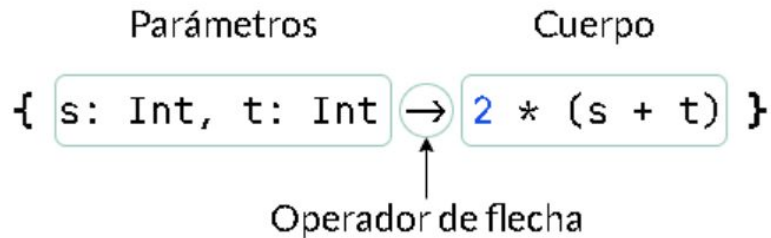
```
{s:Int -> s + 2}
```



# Sintaxis De Lambdas En Kotlin

La sintaxis de un literal lambda va al interior de dos llaves `{ }`. Sus componentes son:

- **Lista de parámetros** – Cada parámetro es una declaración de variable, aunque esta lista es opcional
- **Operador de flecha** `->` – Se omite si no usas lista de parámetros
- **Cuerpo del lambda** – Son las sentencias que van luego del operador de flecha





# Funciones como tipo de datos en Kotlin

En Kotlin, las funciones se consideran construcciones de primera clase. Esto significa que las funciones se pueden tratar como un tipo de datos.

Puedes almacenar funciones en variables, pasarlas a otras funciones como argumentos y mostrarlas desde otras funciones.

Esto en Java se puede simular, pero no se puede replicar exactamente igual.

# Funciones como tipo de datos en Kotlin

Vamos a definir una clase Kotlin con una función que tenga como argumento otra función:

La función se llama myFun y recibe 2 enteros

myFun retorna un double

```
class Areas {  
    fun calcularArea(x: Int, y: Int, myFun : (Int, Int) -> Double) : Double {  
        return myFun(x, y)  
    }  
}
```

calcularArea retorna otro Double e indicamos que será el retorno de myFun

# Funciones como tipo de datos en Kotlin

Ahora, dentro de nuestro Main vamos a utilizar esta clase de cálculo de Áreas

```
val base = 2
val altura = 4

val funCuadrado = fun (x: Int,y: Int): Double {
    return (x * y).toDouble()
}

val myAreas = Areas()
val superficieCuadrado = myAreas.calcularArea(base,
    altura, funCuadrado)
println("La superficie si es un cuadrado es: "
    +superficieCuadrado)
```

Definimos una función anónima y la “almacenamos en una variable” llamada funCuadrado

Pasamos la variable de la función a la clase Áreas en el parámetro de función. La función debe tener los mismos parámetros de entrada y el mismo tipo de retorno



## Funciones como tipo de datos en Kotlin

En realidad no necesito definir una variable de tipo función para pasarla, puedo definir la función cuando la estoy pasando:

```
val base = 2
val altura = 4
val myAreas = Areas()
val superficieCuadrado = myAreas.calcularArea(base, altura, fun (x: Int,y: Int): Double {
    return (x * y).toDouble()
})
println("La superficie si es un cuadrado es: "+superficieCuadrado)
```



## Funciones como tipo de datos en Kotlin

Y según hemos visto la sintaxis de las funciones lamdas, estas dos funciones son equivalentes:

```
fun (x: Int,y: Int): Double {  
    return (x * y).toDouble()  
}
```



```
{ x: Int, y: Int -> (x * y).toDouble() }
```





## Lambda Como Argumento

Por tanto, podemos usar la función lamda como argumento de mi función de calcularArea

```
val base = 2
val altura = 4
val myAreas = Areas()
val superficieCuadrado = myAreas.calcularArea(base, altura,
|   { x: Int, y: Int -> (x * y).toDouble() }
| )
println("La superficie si es un cuadrado es: "+superficieCuadrado)
```



## Lambda Como Argumento

De hecho, aunque funcionaría, tenemos marcado en amarillo la función.... porque cuando la función es el último parámetro podemos, y es lo más habitual, sacarla fuera del parentesis

```
val base = 2
val altura = 4
val myAreas = Areas()
val superficieCuadrado = myAreas.calcularArea(base, altura,
|   { x: Int, y: Int -> (x * y).toDouble() }
| )
println("La superficie si es un cuadrado es: "+superficieCuadrado)
```



## Lambda Como Argumento

Quedaría así:


```
val base = 2
val altura = 4
val myAreas = Areas()
val superficieCuadrado = myAreas.calcularArea(base, altura) { x: Int, y: Int -> (x * y).toDouble() }
println("La superficie si es un cuadrado es: "+superficieCuadrado)
println("La superficie si es un triangulo es: "
    +myAreas.calcularArea(base, altura) { x: Int, y: Int -> ((x * y) / 2).toDouble() })
```



## Omitir Tipo Del Parámetro

Si el contexto permite la inferencia de tipo del parámetro en la lambda, entonces puedes omitirlo del bloque.

```
val superficieCuadrado = myAreas.calcularArea(base, altura) { x, y -> (x * y).toDouble() }
```



No hace falta que indique de qué tipo son x e y puesto que son los parámetros de myFun definidos y por tanto se puede inferir que son Int



## Funciones lamda: ¿que es “it”?

Añadimos a nuestra clase de áreas un función para calcular áreas circulares, sólo recibe un parámetro:

```
class Areas {  
    fun calcularArea(x: Int, y: Int, myFun : (Int, Int) -> Double) : Double {  
        return myFun(x, y)  
    }  
    fun calcularAreaRedonda(x: Int, myFunR : (Int) -> Double) : Double {  
        return myFunR(x)  
    }  
}
```



## Funciones lamda: ¿que es “it”?

Como ya somos “casi” expertos en lamdas, sabemos que podemos utilizar esta función con declaración de función “estándar” o con funciones lamda o flecha.

```
val radio = 3
val funCirculo = fun (x: Int): Double {
    return x*x*Math.PI
}
val areaCirculo = myAreas.calcularAreaRedonda(radio, funCirculo)
val areaCirculo2 = myAreas.calcularAreaRedonda(radio) { x: Int -> x * x * Math.PI }
println("La superficie si es un círculo es: "+areaCirculo)
println("La superficie si es un círculo es: "+areaCirculo2)
```




## Funciones lamda: ¿que es “it”?

Cuando tu lambda usa un único argumento y no piensas cambiar su nombre por cuestiones de legibilidad, puedes usar el identificador `it` (es una palabra “reservada” en Kotlin para este uso).

Esta variable se deduce implícitamente con el tipo inferido por el compilador y puedes referirte a ella como tu parámetro.

```
val areaCirculo3 = myAreas.calcularAreaRedonda(radio) { it -> it * it * Math.PI }
```



it está como grisaceo.... como si sobrara



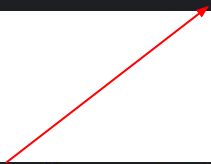
## Funciones lamda: ¿que es “it”?

Porqué “it” es opcional y la mayoría de las veces no lo veremos. Cuando es un sólo parametro podemos usar “it” y saltarnos la definición del parámetro y pasar directamente a su uso

```
val areaCirculo4 = myAreas.calcularAreaRedonda(radio) { it * it * Math.PI }
```

Recordar, veníamos de aquí:

```
val funCirculo = fun (x: Int): Double {  
    return x*x*Math.PI  
}
```







## Funciones lamda: ¿esto se puede hacer en Java?

Se puede, pero con bastante más esfuerzo. Nos definimos una interfaz con un sólo método abstracto (y los que sea que necesitemos no abstractos)

```
public interface AreaIntefaz { 3 usages 1 implementation
    double calcularArea(int x, int y); 1 usage 1 implementation
}
```



## Funciones lamda: ¿esto se puede hacer en Java?

Nos declaramos una clase con un método. Uno de los parámetros de este método será la interfaz (en Java no se pueden definir métodos como parámetros de métodos pero si podemos definir un objeto como parámetro)

```
public class Areas { 2 usages

    public double calcularArea(int x, int y, AreaIntefaz myAreaIntefaz){
        return myAreaIntefaz.calcularArea(x, y);
    }
}
```



## Funciones lamda: ¿esto se puede hacer en Java?

En el momento de usar mi clase Áreas, cuyo método calcularArea utiliza una interfaz, debo definir la función de esta interfaz

```
int base = 2, altura = 4;
Areas myAreas = new Areas();
double superficieCuadrado = myAreas.calcularArea(base, altura, new AreaIntefaz() {
    @Override 1 usage
    public double calcularArea(int x, int y) {
        return x*y;
    }
});
System.out.println("La superficie si es un cuadrado es: "+superficieCuadrado);
```



## Funciones lamda: ¿esto se puede hacer en Java?

Y en este contexto, Java (desde Java 8), soporta la notación de flecha (infiriendo también el tipo de los argumentos)

```
double superficieCuadrado3 = myAreas.calcularArea(base, altura, (x,y) -> { return x*y; });  
System.out.println("La superficie si es un cuadrado es (3): "+superficieCuadrado3);
```

# Documentación

Seguro que hay mucha más, pero a mi me ha resultado especialmente útil la siguiente:

<https://www.develou.com/lambdas-en-kotlin/>

<https://developer.android.com/codelabs/basic-android-kotlin-compose-function-types-and-lambda?hl=es-419#0>



```
new Thread(() -> {  
    doSomething();  
});
```