Reseña:

tiene tres parámetros de entrada, la forma de declararlos es muy fácil el **nombre** de la variable, seguida de **dos puntos** y el **tipo de variable**, aquí si es obligatorio definir el tipo.

Los métodos de Java ahora se conocen como funciones y se declaran con **fun**:

|  |  |
| --- | --- |
| int doble(int x) {     return 2 \* x  } | fun doble(x: Int): Int {     return 2 \* x  } |

Los parámetros y el tipo devuelto se indican usando la notación Pascal. Es posible omitir el tipo que devuelve (también podemos indicar: Unit). Cada variable ha de tener su tipo y no podemos agrupar varias variables cuando son del mismo tipo.

|  |  |  |
| --- | --- | --- |
| void funcion(int p1, p2) {  …  } | fun funcion(p1: Int, p2: Int) {  …  } |  |

Una mejora frente a Java es la posibilidad de indicar un valor por defecto en algún parámetro:

fun rellena(buff: ByteArray, valor: Byte = 0, long: Int = buff.size) {

for (i in 0..long-1) buff[i] = valor

}

La función anterior tiene tres parámetros, pero solo es obligatorio el primero. Como vemos a continuación puede ser llamada de muy diferentes formas.

var b = ByteArray(10) //Creamos un array de bytes con 10 elementos

rellena(b, 99, 5) //Rellena 5 elementos con 99

rellena(b, 99) //Rellena todos elementos con 99

rellena(b) //Rellena todos elementos con 0

rellena(b, long=5) //Rellena 5 elementos con 0

rellena(long=5, buff=b) //Igual que el anterior

Observa como en las dos últimas líneas podemos indicar el nombre de cada parámetro. Es especialmente útil cuando queramos aclarar el significado de cada parámetro, queramos solo indicar algunos o queramos indicarlos en un orden diferente. Como puedes comprobar, la forma de indicar los parámetros en Kotlin es muy flexible. Para conseguir un comportamiento similar en Java sería necesario escribir varias sobrecargas de la función.  
Una función puede tener un número variable de argumentos del mismo tipo:

|  |  |
| --- | --- |
| void imprime(String... cadenas) {     for (String cadena : cadenas)        System.out.println(cadena);  } | fun imprime(vararg cadenas:String){     for (cadena in cadenas)        println(cadena)  } |

Si la función es muy corta podemos escribirla en una línea:

|  |  |
| --- | --- |
| fun doble(x: Int): Int {     return 2\*x  } | fun doble(x: Int) = 2\*x |

En Kotlin, las funciones se pueden declarar en el nivel superior de un archivo, lo que significa que no es necesario crear una clase para crear una función, como en Java, C# o Scala.  
Kotlin admite funciones locales, es decir, una función dentro de otra función:

fun dfs(graph: Graph) {

fun dfs(current: Vertex, visited: Set) {

…

}

dfs(graph.vertices[0], HashSet())

}

Kotlin admite funciones con parámetros genéricos:

|  |  |
| --- | --- |
| void add(T a, Collection c){  c.add(a);  } | fun add(a: T,  c: MutableCollection) {  c.add(a)  } |

Kotlin admite un estilo de programación funcional conocido como recursivi-dad de cola. Esto permite que algunos algoritmos que normalmente se escribi-rían usando bucles se escriban usando una función recursiva, pero sin el ries-go de desbordamiento de la pila. Cuando una función está marcada con el modificador de tailrec y cumple con la forma requerida, el compilador optimiza la recursión, dejando en su lugar una versión rápida y eficiente basada en un bucle:

tailrec fun puntoFijoCoseno (x: Double = 1.0): Double

= if (x == Math.cos(x)) x else puntoFijoCoseno(Math.cos(x))

La función anterior calcula el punto fijo del coseno de forma recursiva. Toda función marcada con tailrec ha de llamarse a sí misma justo al final de la función.