

2. Descubra por qué (casi) todo tiene un valor

1. En Hello.kt , escriba el código main() para asignar el valor de retorno de println() a una variable llamada isUnit e imprímalo. (println() no devuelve explícitamente un valor, por lo que devuelve kotlin.Unit).
2. Ejecuta su programa. El primer println() imprime la cadena "This is an expression". La segunda println() imprime el valor de la primera println() declaración, es decir, kotlin.Unit.

```
Replica-Francisco-Molina > lamda.kt > main
1 fun main()
2 {
3
4     val isUnit = println("Esta es una expresion")
5
6     println(isUnit)
7
8
9 }
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

```
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina> cd "c:\Use
{ kotlinc lamda.kt -include-runtime -d lamda.jar } ; if ($?) { java -jar lamd
Esta es una expresion
kotlin.Unit
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina>
```

3. En su main función () en el archivo Hello.kt , declare una val llamada temperatura e inicialícela en 10.
4. Declare otro val llamado isHot y asigne el valor de retorno de una instrucción if/ else a isHot, como se muestra en el siguiente código. Debido a que es una expresión, puede usar el valor de la if expresión de inmediato.

La salida mostraría:

```
Replica-Francisco-Molina > lamda.kt > main
1  fun main()
2  {
3
4      val temperatura = 10
5      val esCaliente = if(temperatura > 50)true else false
6
7      println(esCaliente)
8
9  }
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

```
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina> c
{ kotlinc lamda.kt -include-runtime -d lamda.jar } ; if ($?) { java
false
```

5. Usa el valor de una expresión en una plantilla de cadena. Agregue algún código en su `main()` para verificar la temperatura y determinar si un pescado es seguro o está demasiado caliente, luego ejecute su programa.

La salida mostraría:

Replica-Francisco-Molina > lamda.kt > main

```
1 fun main()
2 {
3
4     val temperatura = 10
5     val mensaje = "La temperatura del agua esta ${if(temperatura > 50)"Muy caliente" else "OK"}"
6
7     println(mensaje)
8
9 }
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

```
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina> cd "c:\Users\Francisco Molina R\Downloads\Re
{ kotlinc lamda.kt -include-runtime -d lamda.jar } ; if ($?) { java -jar lamda.jar }
La temperatura del agua esta OK
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina> |
```

3. Información sobre las funciones

1. Escribe una función llamada `feedTheFish()` que llame `randomDay()` para obtener un día aleatorio de la semana. Use una plantilla de cuerda para imprimir un food para que los peces se coman ese día. Por ahora, los peces comen la misma comida todos los días.

Replica-Francisco-Molina > lamda.kt > main

```
1 fun main()
2 {
3     alimentarPez()
4 }
5
6 fun alimentarPez()
7 {
8
9     val dia = azarDia()
10    val comida = "Percebe"
11    println("Hoy es ${dia} y los peces comen ${comida}")
12
13 }
```

2. Agregue una `randomDay()` función en `Hello.kt` para elegir un día aleatorio de una matriz y devolverlo.

La `nextInt()` función toma un límite de enteros, que limita el número de `Random()` 0 a 6 para que coincida con la week matriz.

```
Francisco-Molina > lamda.kt > alimentarPez

fun azarDia() : String
{
    val semana = arrayOf ("Lunes","Martes","Miercoles","Jueves",
                           "Viernes","Sabado","Domingo")

    return semana[Random().nextInt(semana.size)]
}
```

3. Las funciones `Random()` y `nextInt()` se definen en `java.util.*`. En la parte superior del archivo, agregue la importación necesaria:

```
tar Terminal Ayuda lamda.kt

lamda.kt X

Replica-Francisco-Molina > lamda.kt > ...
1 import java.util.*
2
```

4. Ejecute su programa y verifique la salida.

```
PROBLEMAS  SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN

PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina> cd "c:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina"
{ kotlinc lamda.kt -include-runtime -d lamda.jar } ; if ($?) { java -jar lamda.jar }
Hoy es Miercoles y los peces comen Percebe
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina> 
```

Paso 2: usa una expresión when

1. En Hello.kt , agregue una función llamada fishFood() que tome un día como un String y devuelva la comida del pez para el día como un String. Úselo when(), para que cada día el pescado reciba un alimento específico. Ejecute su programa varias veces para ver diferentes salidas.

```
fun comidaDia(dia : String): String
{
    var comida = ""

    when(dia)
    {
        "Lunes" -> comida = "flakes"
        "Martes" -> comida = "pellets"
        "Miercoles" -> comida = "redworms"
        "Jueves" -> comida = "granules"
        "Viernes" -> comida = "mosquitoes"
        "Sabado" -> comida = "lettuce"
        "Domingo" -> comida = "plankton"
    }
    return comida
}
```

```
PROBLEMAS  SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN

PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina> cd "c:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina"
{ kotlinc lamda.kt -include-runtime -d lamda.jar } ; if ($?) { java -jar lamda.jar }
Hoy es Miercoles y los peces comen gusanos
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina> |
```

2 . Agregue una rama predeterminada a la when expresión usando else. Para realizar pruebas, para asegurarse de que el programa se utiliza a veces por defecto, elimine las ramas Tuesday y Saturday .Agregue una rama predeterminada a la when expresión usando else. Para realizar pruebas, para asegurarse de que el programa se utiliza a veces por defecto, elimine las ramas Tuesday y Saturday.

```
fun comidaDia(dia : String): String
{
    val comida : String

    when(dia)
    {
        "Lunes" -> comida = "hojuelas"
        "Miercoles" -> comida = "gusanos"
        "Jueves" -> comida = "granos"
        "Viernes" -> comida = "mosquitos"
        "Domingo" -> comida = "plankton"
        else -> comida = "nada"
    }
    return comida
}
```

3. Debido a que cada expresión tiene un valor, puede hacer que este código sea un poco más conciso. Devuelve el valor de la when expresión directamente y elimina la food variable. El valor de la when expresión es el valor de la última expresión de la rama que cumplió la condición.

```
7  
8 fun comidaDia(dia : String): String  
9 {  
10  
11     return when(dia)  
12     {  
13         "Lunes" -> "hojuelas"  
14         "Miercoles" -> "gusanos"  
15         "Jueves" -> "granos"  
16         "Viernes" -> "mosquitos"  
17         "Domingo" -> "plankton"  
18         else -> "nada"  
19     }  
20 }  
21  
22 }
```


4. Explore los valores predeterminados y las funciones compactas

Paso 1: cree un valor predeterminado para un parámetro

1. En Hello.kt , escriba una swim()función con un String parámetro llamado speed que imprima la velocidad del pez. El speed parámetro tiene un valor predeterminado de "fast".
2. Desde la main() función, llame a la swim()función de tres formas. Primero llame a la función usando el valor predeterminado. Luego llame a la función y pase el speed parámetro sin un nombre, luego llame a la función nombrando el speed parámetro.

```
Replica-Francisco-Molina > Nado.kt > nadar
1 fun main()
2 {
3
4     nadar()
5     nadar("lento")
6     nadar("Como tortuga")
7
8
9 }
10
11 fun nadar(velocidad : String = "rapido")
12 {
13
14     println("Esta nadando ${velocidad}")
15
16
17 }
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

```
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina> cd "c:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina" & { kotlinc Nado.kt -include-runtime -d Nado.jar } ; if ($?) { java -jar Nado.jar }
Esta nadando rapido
Esta nadando lento
Esta nadando Como tortuga
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina>
```

Paso 2: agregue los parámetros requeridos

1. En Hello.kt , escribir una shouldChangeWater()función que toma tres parámetros: day, temperature, y un dirty nivel. La función vuelve true si se debe cambiar el agua, lo que sucede si es domingo, si la temperatura es demasiado alta o si el agua está demasiado sucia. Se requiere el día de la semana, pero la temperatura predeterminada es 22 y el nivel de suciedad predeterminado es 20.

```
fun cambiarAgua(day : String, temperature : Int = 32, dirty: Int = 20) : Boolean
{
    return when
    {
        temperature > 30 -> true
        dirty > 30 -> true
        day == "Domingo" -> true
        else -> false
    }
}
```

2. Llame `shouldChangeWater()` desde `feedTheFish()` y suministre el día. El `day` parámetro no tiene un valor predeterminado, por lo que debe especificar un argumento. Los otros dos parámetros de `shouldChangeWater()` tienen valores predeterminados, por lo que no tiene que pasar argumentos para ellos.

```
fun alimentarPez()
{
    val dia = azarDia()
    val comida = comidaDia(dia)
    println("Hoy es ${dia} y los peces comen ${comida}")
    println("Cambiar agua:  ${cambiarAgua(dia)}")
}
```

56

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

```
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina> cd "c:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina"
{ kotlinc lamda.kt -include-runtime -d lamda.jar } ; if ($?) { java -jar lamda.jar }
Hoy es Lunes y los peces comen hojuelas
Cambiar agua: false
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina> |
```

Paso 3: hacer funciones compactas

1. en Hello.kt , agregue funciones compactas para probar las condiciones.

```
PROBLEMAS  SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN

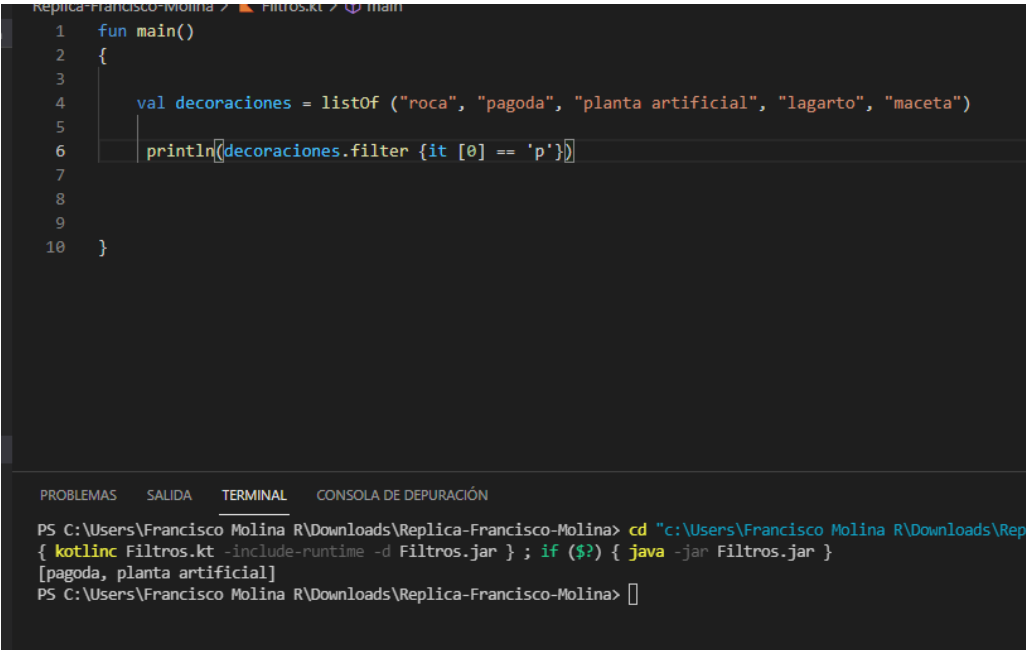
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina> cd "c:\Users\Francisco
{ kotlinc lamda.kt -include-runtime -d lamda.jar } ; if ($?) { java -jar lamda.jar }
Hoy es Lunes y los peces comen hojuelas
Cambiar agua: false
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina> 
```

3. Ejecuta su programa. La salida de println() con shouldChangeWater() debería ser la misma que tenía antes de cambiar al uso de funciones compactas.

```
fun cambiarAgua(day : String, temperature : Int = 22, dirty: Int = 20) : Boolean
{
    return when
    {
        isMuyCaliente(temperature) -> true
        isSucio(dirty) -> true
        isDomingo(day) -> true
        else -> false
    }
}
```

5. Empiece a utilizar filtros

1. En Hello.kt , defina una lista de decoraciones de acuarios en el nivel superior con listOf(). Puede reemplazar el contenido de Hello.kt .
2. Cree una nueva main() función con una línea para imprimir solo las decoraciones que comienzan con la letra 'p'. El código para la condición del filtro está entre llaves {} y se refiere implícitamente a cada elemento a medida que el filtro recorre la lista. Si la expresión regresa true, se incluye el elemento.
3. Ejecute su programa y verá el siguiente resultado en la ventana Ejecutar



The screenshot shows an IDE window with a Kotlin file named Filtros.kt. The code defines a main function that creates a list of aquarium decorations and filters them to only those starting with the letter 'p'. The output of the program is displayed in the terminal window below the code editor.

```
1 fun main()
2 {
3
4     val decoraciones = listOf ("roca", "pagoda", "planta artificial", "lagarto", "maceta")
5
6     println(decoraciones.filter {it [0] == 'p'})
7
8
9
10 }
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

```
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina> cd "c:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina" & { kotlinc Filtros.kt -include-runtime -d Filtros.jar } ; if ($?) { java -jar Filtros.jar }
[pagoda, planta artificial]
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina>
```

Paso 2: comparar filtros ansiosos y perezosos

1. En Hello.kt , cambie su código para asignar la lista filtrada a una variable llamada eager, luego imprímala.
2. Debajo de ese código, evalúe el filtro usando Sequence con asSequence(). Asigne la secuencia a una variable llamada filtered e imprímala.
3. fuerza la evaluación de la secuencia convirtiéndola en List con toList(). Imprime el resultado.
4. Ejecute su programa y observe el resultado.

```
1 fun main()
2 {
3
4     val decoraciones = listOf ("roca", "pagoda", "planta artificial", "lagarto", "maceta")
5
6     println(decoraciones.filter {it [0] == 'p'})
7
8
9     val eager = decoraciones.filter { it [0] == 'p' }
10    println("ansioso: $eager")
11
12    val filtrado = decoraciones.asSequence().filter { it[0] == 'p' }
13    println("filtrado: $filtrado")
14
15    val nuevaLista = filtrado.toList()
16    println("nueva lista: $nuevaLista")
17
18
19 }
```

PROBLEMAS SALIDA **TERMINAL** CONSOLA DE DEPURACIÓN

```
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina> cd "c:\Users\Francisco Molina R\Downloads"
{ kotlinc Filtros.kt -include-runtime -d Filtros.jar } ; if ($?) { java -jar Filtros.jar }
[pagoda, planta artificial]
ansioso: [pagoda, planta artificial]
filtrado: kotlin.sequences.FilteringSequence@3d494fbf
nueva lista: [pagoda, planta artificial]
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina>
```

5. Con la misma decorations lista que la anterior, realice una transformación `map()` que no haga nada y simplemente devuelva el elemento que se pasó. Agregue un `println()` para mostrar cada vez que se acceda a un elemento y asigne la secuencia a una variable llamada `lazyMap`.

```
17
18     val MapaPerezoso = decoraciones.asSequence().map{
19
20         println("Acceso: ${it}")
21         it
22     }
23
24
```

6. Imprima `lazyMap`, imprima el primer elemento de `lazyMap` uso `first()` e imprima `lazyMap` convertido a `List`.

```
println("Con Pereza: ${MapaPerezoso}")
println("-----")
println("Primero: ${MapaPerezoso.first()}")
println("-----")
println("Todo: ${MapaPerezoso.toList()}")
```

7. Ejecute su programa y observe el resultado. La impresión lazyMap solo imprime una referencia al Sequence—el interior println() no se llama. Al imprimir el primer elemento se accede solo al primer elemento. La conversión de Sequence a List accede a todos los elementos.

```
Con Pereza: kotlin.sequences.TransformingSequence@17a7cec2
-----
Acceso: roca
Primero: roca
-----
Acceso: roca
Acceso: pagoda
Acceso: planta artificial
Acceso: lagarto
Acceso: maceta
Todo: [roca, pagoda, planta artificial, lagarto, maceta]
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina>
```


8. Cree un nuevo Sequence con el filtro original antes de aplicarlo map. Imprime ese resultado.
9. Ejecute su programa y observe la salida adicional. Al igual que con la obtención del primer elemento, el interior println() solo se llama para los elementos a los que se accede.

```
31     val MapaPerezoso2 = decoraciones.asSequence().filter { it[0] == 'p' }.map {
32
33         println("Acceso: ${it}")
34         it
35     }
36     println("-----")
37     println("Filtrado: ${MapaPerezoso2.toList()}")
38
39
40
41 }
42
43
44
45
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

Primero: roca

Acceso: roca
Acceso: pagoda
Acceso: planta artificial
Acceso: lagarto
Acceso: maceta
Todo: [roca, pagoda, planta artificial, lagarto, maceta]

Acceso: pagoda
Acceso: planta artificial
Filtrado: [pagoda, planta artificial]
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina>

ads: Updating class path

10. Otra función de transformación útil para las colecciones de Kotlin es `flatten()`. Esta función crea una lista a partir de una colección de colecciones, por ejemplo, una matriz de matrices o una lista de listas.
11. Crea una lista de listas. Luego aplique la función `flatten()` para transformar todas las listas en una lista. Imprime el resultado.

```
39
40 val misdeportes = listOf("basketball", "Pesca", "running")
41 val misJugadores = listOf("LeBron James", "Ernest Hemingway", "Usain Bolt")
42 val misCiudades = listOf("Los Angeles", "Chicago", "Jamaica")
43 val miLista = listOf(misdeportes, misJugadores, misCiudades)
44 println("-----")
45 println("Flat: ${miLista.flatten()}")
46
47
48
49
50
51
52
53
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

Acceso: roca
Acceso: pagoda
Acceso: planta artificial
Acceso: lagarto
Acceso: maceta
Todo: [roca, pagoda, planta artificial, lagarto, maceta]

Acceso: pagoda
Acceso: planta artificial
Filtrado: [pagoda, planta artificial]

Flat: [basketball, Pesca, running, LeBron James, Ernest Hemingway, Usain Bolt, Los Angeles, Chicago, Jamaica]
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina>

6. Empiece a utilizar lambdas y funciones de orden superior

Paso 1: aprenda sobre las lambdas

1. Al igual que las funciones con nombre, las lambdas pueden tener parámetros. Para lambdas, los parámetros (y sus tipos, si es necesario) van a la izquierda de lo que se llama una flecha de función `->`. El código a ejecutar va a la derecha de la flecha de la función. Una vez que la lambda está asignada a una variable, puede llamarla como una función.

```
1
2 fun main(){
3
4     var nivelSuciedad = 20
5     val filtroAgua = {suciedad : Int -> suciedad / 2}
6     println(filtroAgua(nivelSuciedad))
7
8 }
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

```
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina> cd "c:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina\" ; if ($?) { kotlinc Lambdas.kt -include-runtime -d Lambdas.jar } ; if ($?) { java -jar Lambdas.jar }
```

10

```
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina> 
```

2. La sintaxis de Kotlin para los tipos de funciones está estrechamente relacionada con su sintaxis para lambdas. Utilice esta sintaxis para declarar limpiamente una variable que tenga una función:

```
val filtroAgua: (Int) -> Int = {suciedad -> suciedad / 2}
```

Paso 2: crea una función de orden superior

1. Escribe una función de orden superior. Aquí hay un ejemplo básico, una función que toma dos argumentos. El primer argumento es un número entero. El segundo argumento es una función que toma un número entero y devuelve un número entero. Pruébalo en el REPL.
2. Para llamar a esta función, pásale un número entero y una función.

```
1
2 fun main() {
3
4
5
6     val filtroAgua: (Int) -> Int = {suciedad -> suciedad / 2}
7
8
9
10    println(sobreSuciedad(30,filtroAgua))
11
12 }
13
14 fun sobreSuciedad(suciedad: Int, operacion: (Int) -> Int) : Int
15 {
16     return operacion(suciedad)
17 }
18
19
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

```
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina> cd "c:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina\" ; if ($?) { kotlinc Lambdas.kt -include-runtime -d Lambdas.jar } ; if ($?) { java -jar Lambdas.jar }
15
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina> 
```

3. Intente pasar una función con nombre regular a update Dirty().

```
1
2 fun main() {
3
4     val filtroAgua: (Int) -> Int = {suciedad -> suciedad / 2}
5
6     println(sobreSuciedad(30,filtroAgua))
7     println(sobreSuciedad(15, ::incrementaSuciedad))
8
9 }
10
11 fun sobreSuciedad(suciedad: Int, operacion: (Int) -> Int) : Int
12 {
13     return operacion(suciedad)
14 }
15
16 fun incrementaSuciedad(start: Int) = start + 1
17
18
19
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina> cd "c:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina\" ; if (\$?) { kotlinc Lambdas.kt -include-runtime -d Lambdas.jar } ; if (\$?) { java -jar Lambdas.jar }
15
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina> cd "c:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina\" ; if (\$?) { kotlinc Lambdas.kt -include-runtime -d Lambdas.jar } ; if (\$?) { java -jar Lambdas.jar }
15
16
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina>

utur Terminal Ayuda Lambdas.kt - Downloads - Visual Studio Code

Lambdas.kt 1 X

Replica-Francisco-Molina > Lambdas.kt > main

```
1
2 fun main(){
3
4     val filtroAgua: (Int) -> Int = {suciedad -> suciedad / 2}
5
6     println(sobreSuciedad(30,filtroAgua))
7     println(sobreSuciedad(15, ::incrementaSuciedad))
8
9     var nivelSuciedad = 19
10    nivelSuciedad = sobreSuciedad(nivelSuciedad) { nivelSuciedad -> nivelSuciedad + 23 }
11    println(nivelSuciedad)
12
13 }
14
15 fun sobreSuciedad(suciedad: Int, operacion: (Int) -> Int) : Int
16 {
17     return operacion(suciedad)
18 }
19
```

PROBLEMAS 1 SALIDA TERMINAL CONSOLA DE DEPURACIÓN

PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina> cd "c:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina\" ; if (\$?) { kotlinc Lambdas.kt -include-runtime -d Lambdas.jar } ; if (\$?) { java -jar Lambdas.jar }
Lambdas.kt:10:50: warning: name shadowed: nivelSuciedad
 nivelSuciedad = sobreSuciedad(nivelSuciedad) { nivelSuciedad -> nivelSuciedad + 23 }
 ^
15
16
42
PS C:\Users\Francisco Molina R\Downloads\Replica-Francisco-Molina>

All Badges



Lesson 2: Functions

8 sept 2021

Completed Lesson 2: Functions

Insignia del primer
camino de
aprendizaje v

