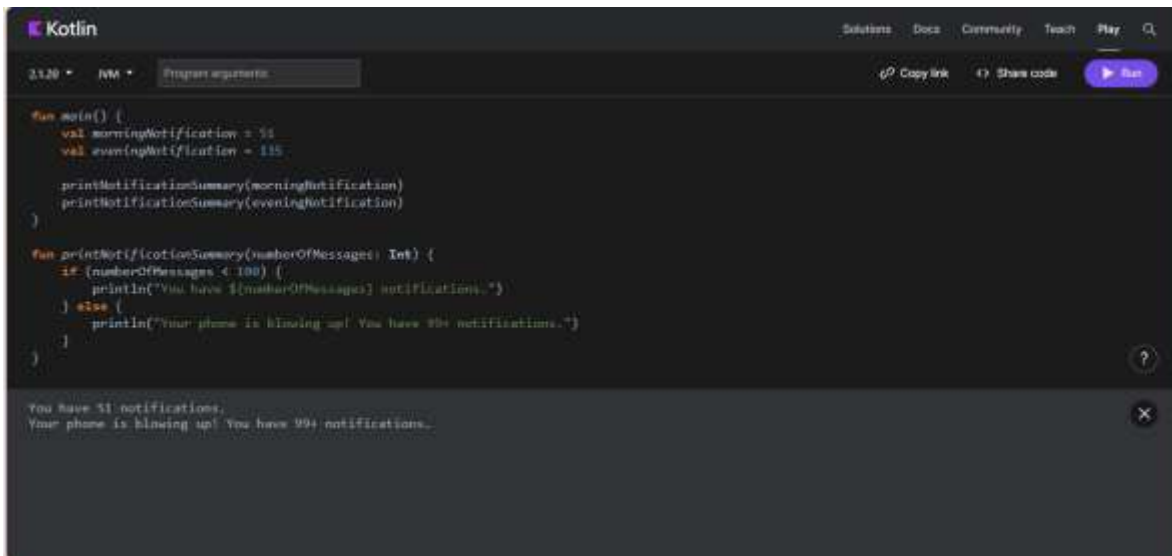


## Programa 1:

A screenshot of the Kotlin Playground web application. The interface has a dark theme. At the top, there's a header with the Kotlin logo, version 1.1.20, and tabs for Solutions, Docs, Community, Teach, and Play. Below the header, there's a text input field with "Program arguments:" and buttons for "Copy link", "Share code", and "Run". The main area contains Kotlin code for a notification summary program. The code defines a main function and a helper function printNotificationSummary. The output at the bottom shows the results of running the code: "You have 51 notifications." and "Your phone is blowing up! You have 99+ notifications.".

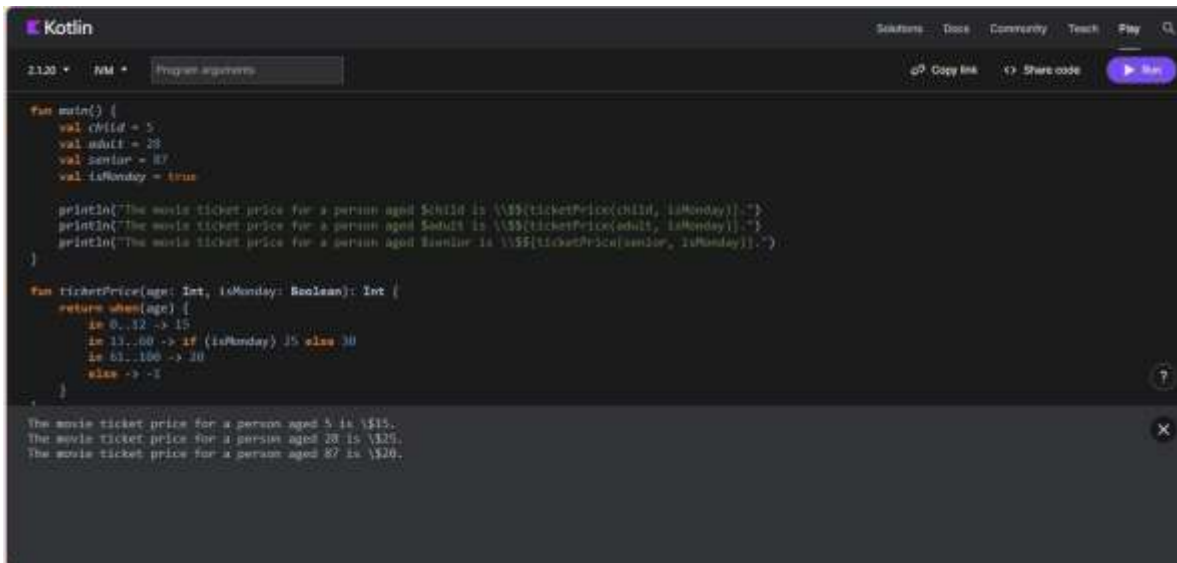
```
fun main() {  
    val morningNotification = 51  
    val eveningNotification = 135  
  
    printNotificationSummary(morningNotification)  
    printNotificationSummary(eveningNotification)  
}  
  
fun printNotificationSummary(numberOfMessages: Int) {  
    if (numberOfMessages < 100) {  
        println("You have ${numberOfMessages} notifications.")  
    } else {  
        println("Your phone is blowing up! You have 99+ notifications.")  
    }  
}
```

You have 51 notifications.  
Your phone is blowing up! You have 99+ notifications.

```
fun main() {  
    val morningNotification = 51  
    val eveningNotification = 135  
  
    printNotificationSummary(morningNotification)  
    printNotificationSummary(eveningNotification)  
}  
  
fun printNotificationSummary(numberOfMessages: Int) {  
    if (numberOfMessages < 100) {  
        println("You have ${numberOfMessages} notifications.")  
    } else {  
        println("Your phone is blowing up! You have 99+ notifications.")  
    }  
}
```

**Documentación:** Este ejercicio trabaja con condicionales. Si el número de mensajes es menor a 100, se imprime el número exacto; si es mayor o igual, se muestra un mensaje genérico. La lógica se encapsula en la función `printNotificationSummary()`.

## Programa 2:



```
fun main() {
    val child = 5
    val adult = 28
    val senior = 87
    val isMonday = true

    println("The movie ticket price for a person aged $child is \${ticketPrice(child, isMonday)}.")
    println("The movie ticket price for a person aged $adult is \${ticketPrice(adult, isMonday)}.")
    println("The movie ticket price for a person aged $senior is \${ticketPrice(senior, isMonday)}.")
}

fun ticketPrice(age: Int, isMonday: Boolean): Int {
    return when(age) {
        in 0..12 -> 15
        in 13..60 -> if (isMonday) 25 else 30
        in 61..100 -> 30
        else -1
    }
}
```

The movie ticket price for a person aged 5 is \$15.  
The movie ticket price for a person aged 28 is \$30.  
The movie ticket price for a person aged 87 is \$30.

```
fun main() {
    val child = 5
    val adult = 28
    val senior = 87
    val isMonday = true

    println("The movie ticket price for a person aged $child is \${ticketPrice(child, isMonday)}.")
    println("The movie ticket price for a person aged $adult is \${ticketPrice(adult, isMonday)}.")
    println("The movie ticket price for a person aged $senior is \${ticketPrice(senior, isMonday)}.")
}
```

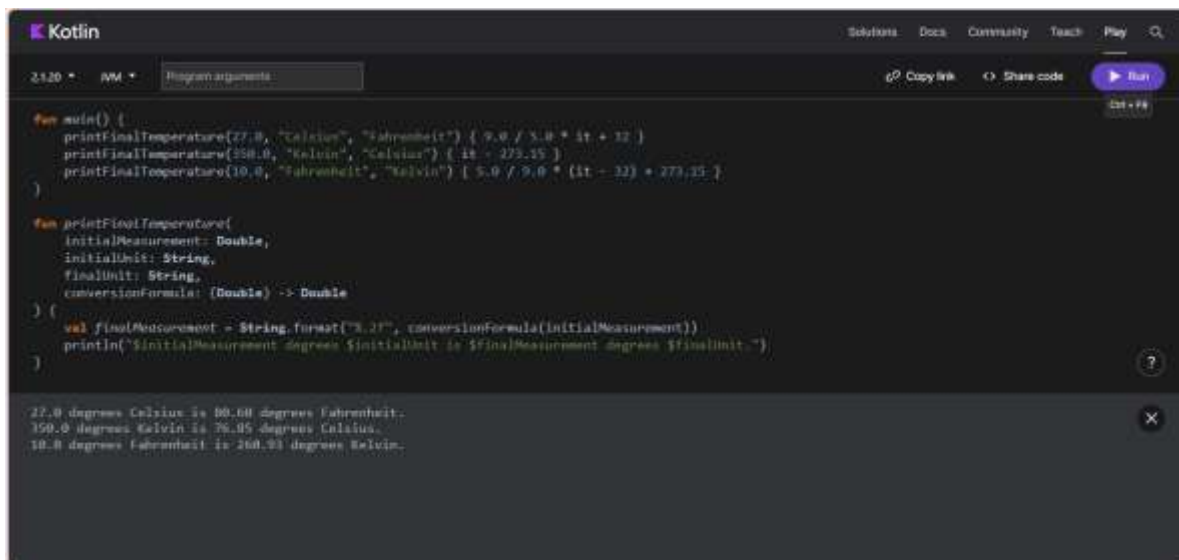
```

fun ticketPrice(age: Int, isMonday: Boolean): Int {
    return when(age) {
        in 0..12 -> 15
        in 13..60 -> if (isMonday) 25 else 30
        in 61..100 -> 20
        else -> -1
    }
}

```

**Documentación:** Se calcula el precio de una entrada de cine usando `when`, según la edad y si es lunes. Las condiciones cambian el precio y el resultado se imprime usando interpolación.

### Programa 3:



```

Kotlin
29.20 * VM * Program arguments
Copy link Share code Run
29.20 * VM * Program arguments
Copy link Share code Run

fun main() {
    printFinalTemperature(27.0, "Celsius", "Fahrenheit") { 9.0 / 5.0 * it + 32 }
    printFinalTemperature(350.0, "Kelvin", "Celsius") { it - 273.15 }
    printFinalTemperature(10.0, "Fahrenheit", "Kelvin") { 5.0 / 9.0 * (it - 32) + 273.15 }
}

fun printFinalTemperature(
    initialMeasurement: Double,
    initialUnit: String,
    finalUnit: String,
    conversionFormula: (Double) -> Double
) {
    val finalMeasurement = String.format("%.2f", conversionFormula(initialMeasurement))
    println("$initialMeasurement degrees $initialUnit is $finalMeasurement degrees $finalUnit.")
}

27.0 degrees Celsius is 80.60 degrees Fahrenheit.
350.0 degrees Kelvin is 76.85 degrees Celsius.
10.0 degrees Fahrenheit is 260.93 degrees Kelvin.

```

```

fun main() {
    printFinalTemperature(27.0, "Celsius", "Fahrenheit") { 9.0 / 5.0 * it + 32 }
    printFinalTemperature(350.0, "Kelvin", "Celsius") { it - 273.15 }
}

```

```

    printFinalTemperature(10.0, "Fahrenheit", "Kelvin") { 5.0 / 9.0 * (it - 32) +
273.15 }
}

```

```

fun printFinalTemperature(
    initialMeasurement: Double,
    initialUnit: String,
    finalUnit: String,
    conversionFormula: (Double) -> Double
) {
    val          finalMeasurement          =          String.format("%.2f",
conversionFormula(initialMeasurement))

    println("$initialMeasurement degrees $initialUnit is $finalMeasurement
degrees $finalUnit.")
}

```

**Documentación:** Este programa convierte entre Celsius, Fahrenheit y Kelvin usando funciones lambda. La función `printFinalTemperature()` recibe una fórmula como parámetro y la aplica. Esto demuestra el uso de funciones de orden superior.

**Programa 4:**

A screenshot of the Kotlin Playground web interface. The top bar shows the Kotlin logo and navigation links: Solutions, Docs, Community, Teach, and Play. Below the bar, there's a tab labeled 'Program arguments' and buttons for 'Copy link', 'Share code', and 'Run'. The main editor area contains Kotlin code. The code defines a 'Song' class with properties 'title', 'artist', 'yearPublished', and 'playCount'. It also has a computed property 'isPopular' that returns true if 'playCount' is greater than or equal to 1000. A 'printDescription()' function is defined to print the song details. In the 'main()' function, a 'keniaSong' object is created with the title 'Cambios', artist 'Kenia OS', year 2022, and play count 1,000,000. The program then calls 'printDescription()' and checks 'isPopular'. The output at the bottom shows the description of the song and the boolean result 'true'.

```
fun main() {  
    val keniaSong = Song("Cambios", "Kenia OS", 2022, 1_000_000)  
    keniaSong.printDescription()  
    println(keniaSong.isPopular)  
}  
  
class Song(  
    val title: String,  
    val artist: String,  
    val yearPublished: Int,  
    val playCount: Int  
) {  
    val isPopular: Boolean  
        get() = playCount >= 1000  
  
    fun printDescription() {  
        println("Title, performed by $artist, was released in $yearPublished.")  
    }  
}
```

Cambios, performed by Kenya OS, was released in 2022.  
true

```
fun main() {  
    val keniaSong = Song("Cambios", "Kenia OS", 2022, 1_000_000)  
    keniaSong.printDescription()  
    println(keniaSong.isPopular)  
}
```

```
class Song(  
    val title: String,  
    val artist: String,  
    val yearPublished: Int,  
    val playCount: Int  
) {  
    val isPopular: Boolean  
        get() = playCount >= 1000  
  
    fun printDescription() {
```

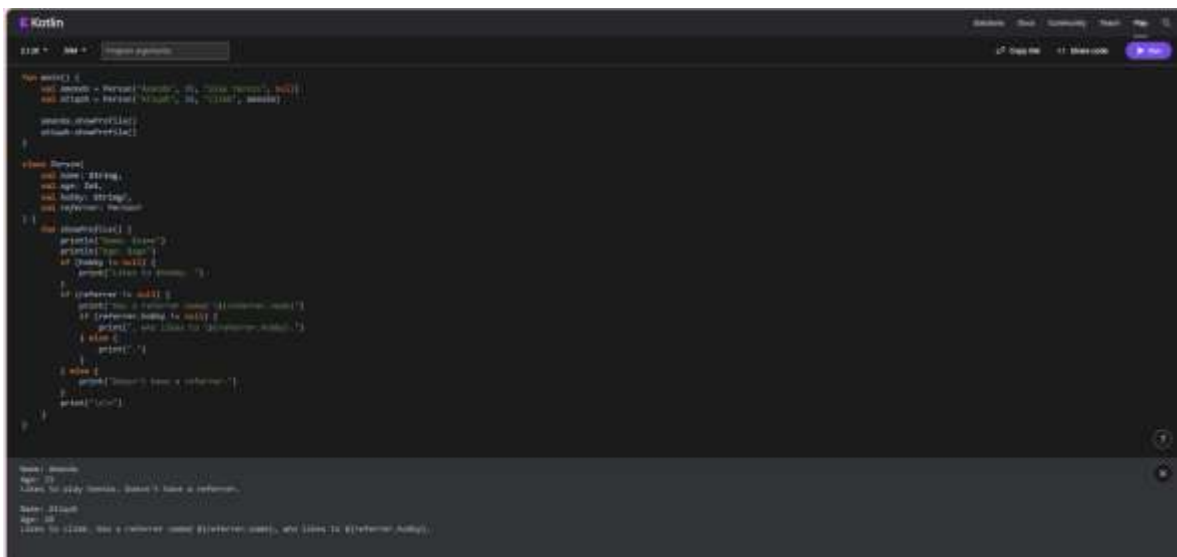
```

        println("$title, performed by $artist, was released in $yearPublished.")
    }
}

```

**Documentación:** Se define una clase `Song` con propiedades como título, artista y número de reproducciones. Contiene un método `printDescription()` y una propiedad calculada `isPopular`. Se instancia un objeto y se verifica si es popular.

### Programa 5:



```

fun main() {
    val amanda = Person("Amanda", 33, "play tennis", null)
    val atiqah = Person("Atiqah", 28, "climb", amanda)

    amanda.showProfile()
    atiqah.showProfile()
}

```

```

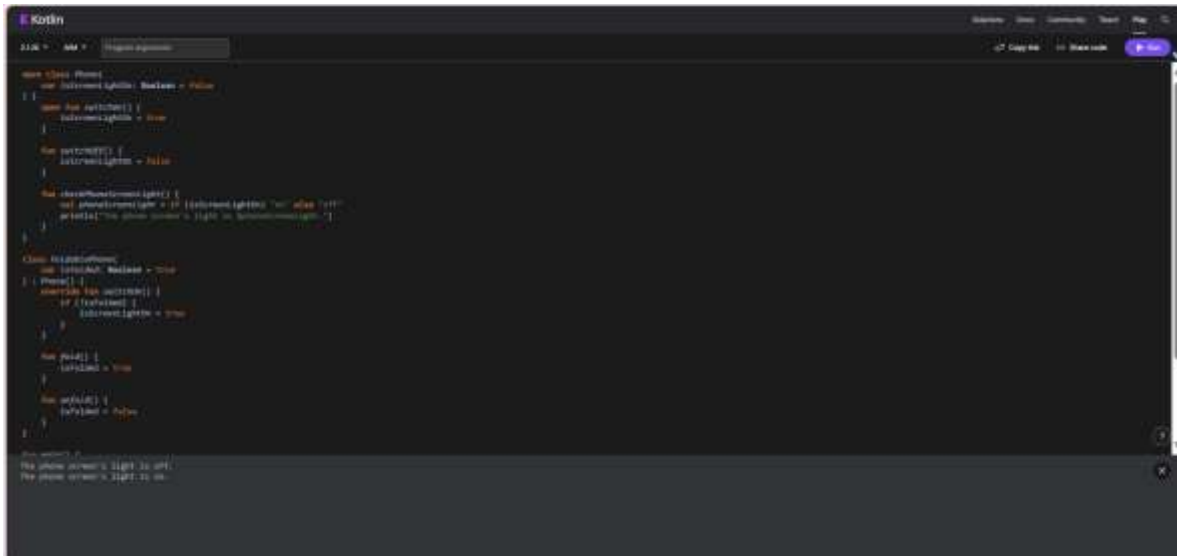
class Person(

```

```
val name: String,
val age: Int,
val hobby: String?,
val referrer: Person?
) {
    fun showProfile() {
        println("Name: $name")
        println("Age: $age")
        if (hobby != null) {
            print("Likes to $hobby. ")
        }
        if (referrer != null) {
            print("Has a referrer named \${referrer.name}")
            if (referrer.hobby != null) {
                print(", who likes to \${referrer.hobby}.")
            } else {
                print(".")
            }
        } else {
            print("Doesn't have a referrer.")
        }
        print("\n\n")
    }
}
```

**Documentación:** La clase `Person` representa a una persona con un nombre, edad, hobby y un referente. La función `showProfile()` imprime el perfil. Se demuestra el uso de objetos relacionados y valores nulos.

## Programa 6:



```
open class Person(  
    var isScreenLightOn: Boolean = false  
) {  
    open fun switchOn() {  
        isScreenLightOn = true  
    }  
  
    fun switchOff() {  
        isScreenLightOn = false  
    }  
  
    fun checkPhoneScreenLight() {  
        val phoneScreenLight = if (isScreenLightOn) "on" else "off"
```



```
        println("The phone screen's light is $phoneScreenLight.")
    }
}
```

```
class FoldablePhone(
    var isFolded: Boolean = true
) : Phone() {
    override fun switchOn() {
        if (!isFolded) {
            isScreenLightOn = true
        }
    }
}
```

```
fun fold() {
    isFolded = true
}
```

```
fun unfold() {
    isFolded = false
}
}
```

```
fun main() {
    val newFoldablePhone = FoldablePhone()
```

```

newFoldablePhone.switchOn()

newFoldablePhone.checkPhoneScreenLight()


newFoldablePhone.unfold()

newFoldablePhone.switchOn()

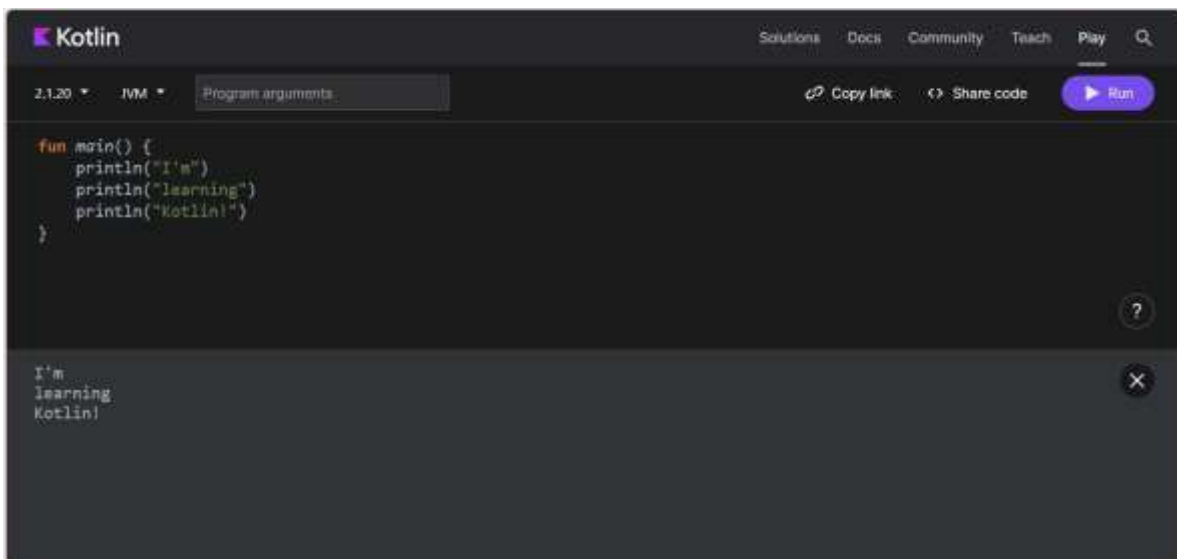
newFoldablePhone.checkPhoneScreenLight()

}

```

**Documentación:** Este ejercicio muestra herencia en Kotlin. La clase `Phone` tiene funciones para encender/apagar la luz de pantalla. `FoldablePhone` hereda de `Phone` y añade lógica adicional: solo enciende si el teléfono está desplegado.

## Programa 7:



The screenshot shows the Kotlin Playground interface. At the top, there's a header with the Kotlin logo and navigation links: Solutions, Docs, Community, Teach, Play, and a search icon. Below the header, there's a toolbar with a version dropdown (2.1.20), a JVM dropdown, a text input for 'Program arguments', and buttons for 'Copy link', 'Share code', and 'Run'. The main editor area contains the following Kotlin code:

```

fun main() {
    println("I'm")
    println("learning")
    println("Kotlin!")
}

```

Below the editor, the output console shows the results of the program execution:

```

I'm
learning
Kotlin!

```

```

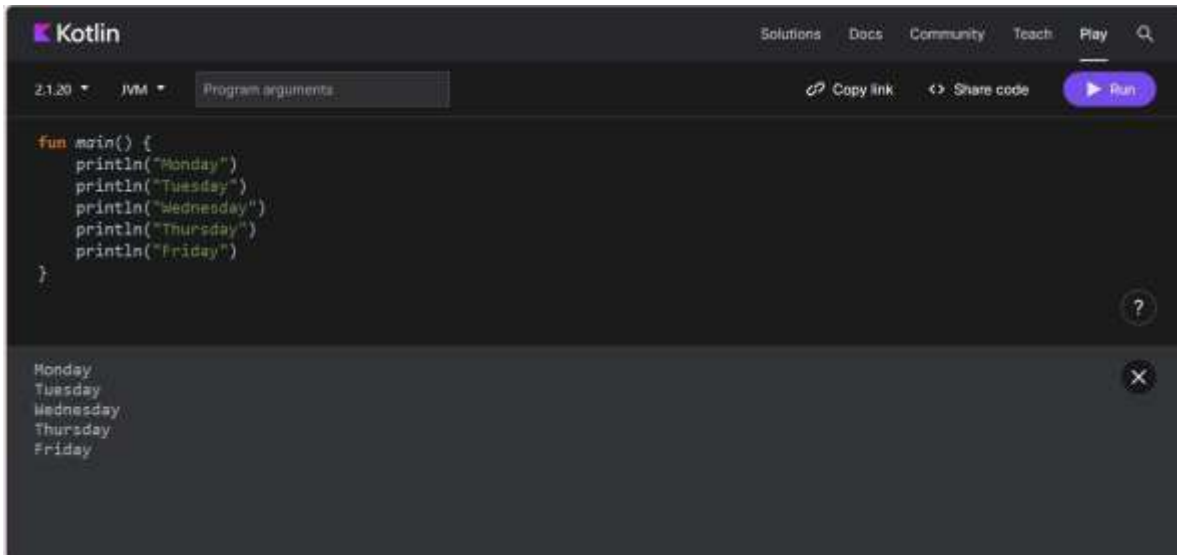
fun main() {
    println("I'm")
    println("learning")
    println("Kotlin!")
}

```

```
}
```

**Documentación:** Este programa usa la función ``println()`` para imprimir tres líneas de texto. Cada llamada a ``println()`` muestra un mensaje seguido de un salto de línea. Es útil para introducir la estructura básica de un programa en Kotlin.

### Programa 8:

The image shows the Kotlin Playground web interface. At the top, there's a header with the Kotlin logo and navigation links: Solutions, Docs, Community, Teach, Play, and a search icon. Below the header, there's a toolbar with '2.1.20', 'JVM', a 'Program arguments' input field, 'Copy link', 'Share code', and a 'Run' button. The main area contains a Kotlin code snippet: 

```
fun main() {  
    println("Monday")  
    println("Tuesday")  
    println("Wednesday")  
    println("Thursday")  
    println("Friday")  
}
```

 To the right of the code is a help icon (?). Below the code editor is a console output area showing the result of running the code: 

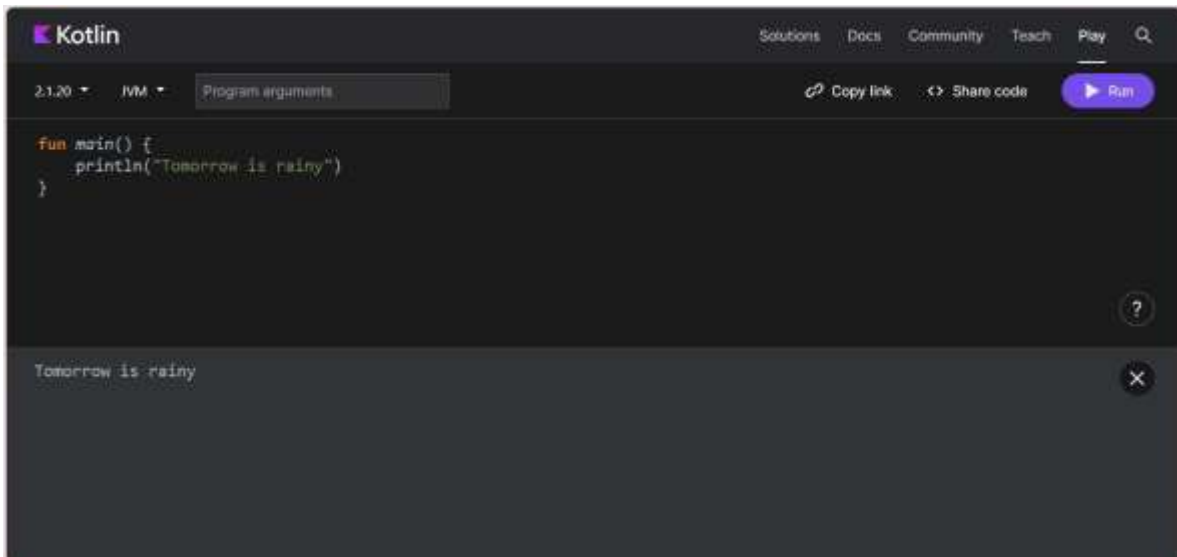
```
Monday  
Tuesday  
Wednesday  
Thursday  
Friday
```

 To the right of the console is a close icon (X).

```
fun main() {  
    println("Monday")  
    println("Tuesday")  
    println("Wednesday")  
    println("Thursday")  
    println("Friday")  
}
```

**Documentación:** Este código imprime los días laborales de lunes a viernes en consola. Se practica la repetición de llamadas a ``println()`` para mostrar texto secuencial.

### Programa 9:



The screenshot shows the Kotlin Playground web interface. At the top, there's a header with the Kotlin logo and navigation links: Solutions, Docs, Community, Teach, and Play. Below the header, there's a toolbar with version (2.1.20), JVM, and a text input field containing 'Program arguments'. To the right of the input field are buttons for 'Copy link', 'Share code', and 'Run'. The main area is a code editor with the following Kotlin code:

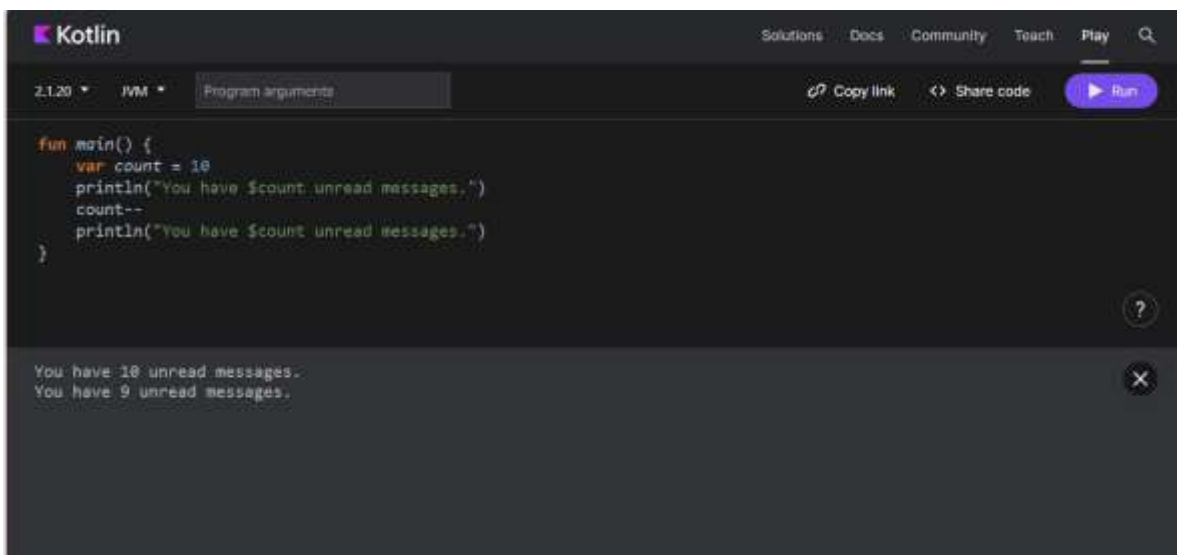
```
fun main() {  
    println("Tomorrow is rainy")  
}
```

Below the code editor, there's a console output area showing the result of the program: 'Tomorrow is rainy'.

```
fun main() {  
    println("Tomorrow is rainy")  
}
```

**Documentación:** El programa muestra una predicción del clima usando una sola línea. Demuestra cómo un programa puede ser tan simple como una única salida de texto.

## Programa 10:



The screenshot shows the Kotlin Playground web interface. At the top, there's a header with the Kotlin logo and navigation links: Solutions, Docs, Community, Teach, and Play. Below the header, there's a toolbar with version (2.1.20), JVM, and a text input field containing 'Program arguments'. To the right of the input field are buttons for 'Copy link', 'Share code', and 'Run'. The main area is a code editor with the following Kotlin code:

```
fun main() {  
    var count = 10  
    println("You have $count unread messages.")  
    count--  
    println("You have $count unread messages.")  
}
```

Below the code editor, there's a console output area showing the result of the program: 'You have 10 unread messages.' and 'You have 9 unread messages.'

```
fun main() {  
    var count = 10
```

```

println("You have $count unread messages.")

count--

println("You have $count unread messages.")
}

```

**Documentación:** Este código utiliza una variable mutable `count` para simular la cantidad de mensajes. Primero muestra 10, luego se decrementa usando `count--` y vuelve a imprimir el nuevo valor. Sirve para entender variables, mutabilidad y operadores.

### Programa 11:

The screenshot shows the Kotlin Playground interface. At the top, there's a header with the Kotlin logo and navigation links: Solutions, Docs, Community, Teach, Play, and a search icon. Below the header, there's a toolbar with a version selector (2.1.20), a target selector (JVM), a text input field for program arguments, and buttons for Copy link, Share code, and Run. The main area contains the following Kotlin code:

```

fun birthdayGreeting(name: String, age: Int): String {
    val nameGreeting = "Happy Birthday, $name!"
    val ageGreeting = "You are now $age years old!"
    return "$nameGreeting\n$ageGreeting"
}

fun main() {
    println(birthdayGreeting("Rover", 5))
    println(birthdayGreeting("Rex", 2))
}

```

Below the code, the output is displayed:

```

Happy Birthday, Rover!
You are now 5 years old!
Happy Birthday, Rex!
You are now 2 years old!

```

```

fun birthdayGreeting(name: String, age: Int): String {
    val nameGreeting = "Happy Birthday, $name!"
    val ageGreeting = "You are now $age years old!"
    return "$nameGreeting\n$ageGreeting"
}

```

```

fun main() {
    println(birthdayGreeting("Rover", 5))
}

```

```
println(birthdayGreeting("Rex", 2))  
}
```

**Documentación:** Define una función `birthdayGreeting()` que recibe un nombre y una edad, y devuelve un mensaje personalizado. Se llama dos veces desde `main()` y se imprime la cadena devuelta. Reforzamos uso de funciones, parámetros y retorno.