



GOBIERNO DE
MÉXICO

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO



TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

Carrera: Ingeniería en Sistemas Computacionales.

Materia: Programación Nativa Para Móviles

Alumnos:

Yañez Herrera Karina Aileen

Zavala Osorio Camilo Alexander

Números de control:

21070415

21070336

Profesor: Jorge Peralta Escobar

Hora: 2:00 pm – 3:00 pm

Semestre: Enero - Junio 2025.

D: Este es un ejemplo básico en Kotlin que demuestra cómo imprimir texto en la consola utilizando la función `println()`.

Descripción General: Este es un ejemplo básico en Kotlin que demuestra cómo imprimir texto en la consola utilizando la función `println()`.

```
// Leccion 1

/*fun main() { // Define la función principal, el punto de entrada del programa.

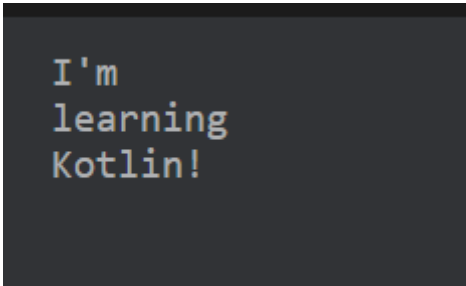
    println("I'm") // Imprime "I'm" en la consola, seguido de una nueva línea.

    println("learning") // Imprime "learning" en la consola, seguido de una nueva
línea.

    println("Kotlin!") // Imprime "Kotlin!" en la consola, seguido de una nueva línea.

}*/
```

Ejecución:



```
I'm
learning
Kotlin!
```

Descripción General: Este programa escrito en Kotlin imprime los días laborales de la semana (de lunes a viernes) en la consola, uno por línea. Utiliza la función principal `main()` y la función `println()` para mostrar cada día.

```
// Leccion 1

/*fun main() { // Define la función principal, el punto de entrada del programa.

    println("Monday") // Imprime "Monday" en la consola, seguido de una nueva
línea.

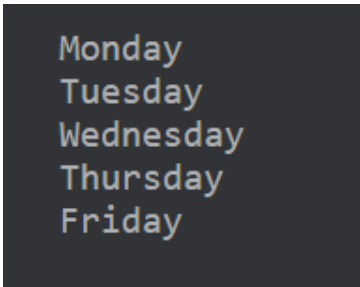
    println("Tuesday") // Imprime "Tuesday" en la consola, seguido de una nueva
línea.

    println("Wednesday") // Imprime "Wednesday" en la consola, seguido de una
nueva línea.
```

```
println("Thursday") // Imprime "Thursday" en la consola, seguido de una nueva línea.
```

```
println("Friday") // Imprime "Friday" en la consola, seguido de una nueva línea.  
}*/
```

Ejecucion:



```
Monday  
Tuesday  
Wednesday  
Thursday  
Friday
```

Descripción General: Este programa simple en Kotlin imprime una predicción del clima: "Tomorrow is rainy" (Mañana va a llover). Es útil para practicar la estructura básica de un programa con una sola salida en consola.

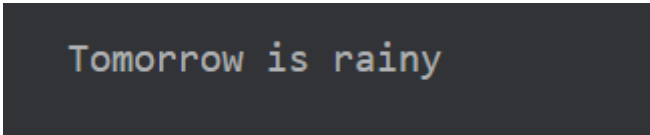
```
// Leccion 1
```

```
/*fun main() { // Define la función principal, el punto de entrada del programa.
```

```
    println("Tomorrow is rainy") // Imprime "Tomorrow is rainy" en la consola,  
    seguido de una nueva línea.
```

```
}*/
```

Ejecucion:



```
Tomorrow is rainy
```

Descripción General: Este programa simula un sistema de notificación de mensajes no leídos en una bandeja de entrada. Utiliza una variable para contar

los mensajes y demuestra cómo modificarla e imprimir su valor con interpolación de cadenas.

```
// Leccion 2
```

```
/*
```

```
* Este programa muestra la cantidad de mensajes
```

```
* en la bandeja de entrada del usuario.
```

```
*/
```

```
/*fun main() { // Define la función principal, el punto de entrada del programa.
```

```
    // Crear una variable para la cantidad de mensajes no leídos.
```

```
    var count = 10 // Declara una variable mutable 'count' y la inicializa en 10.
```

```
    println("You have $count unread messages.") // Imprime la cantidad de  
    mensajes no leídos usando interpolación de cadenas.
```

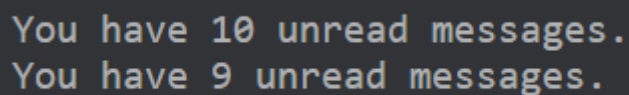
```
    // Disminuir la cantidad de mensajes en 1.
```

```
    count-- // Decrementa el valor de 'count' en 1 (count se convierte en 9).
```

```
    println("You have $count unread messages.") // Imprime la cantidad actualizada  
    de mensajes no leídos.
```

```
}*/
```

Ejecucion:



```
You have 10 unread messages.  
You have 9 unread messages.
```

Descripción General: Este programa define una función que genera un mensaje de cumpleaños personalizado usando el nombre y la edad proporcionados. Luego, se llama a esta función desde main() para imprimir saludos de cumpleaños para dos destinatarios.

```
// Leccion 4
```

```
/*fun birthdayGreeting(name: String, age: Int): String { // Define una función que  
    toma un nombre y edad, y devuelve una cadena.
```

```

    val nameGreeting = "Happy Birthday, $name!" // Crea una constante
'ageGreeting' con un mensaje de cumpleaños usando el nombre
proporcionado.

    val ageGreeting = "You are now $age years old!" // Crea una constante
'ageGreeting' con un mensaje de edad usando la edad proporcionada.

    return "$nameGreeting\n$ageGreeting" // Devuelve ambos mensajes
combinados, con un carácter de nueva línea entre ellos.

}

fun main() { // Define la función principal, el punto de entrada del programa.

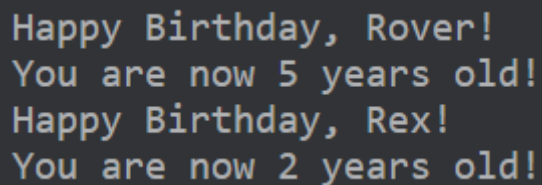
    println(birthdayGreeting("Rover", 5)) // Llama a birthdayGreeting con "Rover" y 5,
e imprime el resultado.

    println(birthdayGreeting("Rex", 2)) // Llama a birthdayGreeting con "Rex" y 2, e
imprime el resultado.

}*/

```

Ejecucion:



```

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

```

Descripción General: Este programa simula la visualización de notificaciones en un teléfono. Dependiendo de la cantidad de mensajes, muestra el número exacto o un mensaje genérico indicando que hay más de 99 notificaciones.

```
// Impresion de mensajes
```

```
/*fun main() { // Define la función principal, el punto de entrada del programa.
```

```
    val morningNotification = 51 // Declara una constante 'morningNotification' con
valor 51.
```

```
    val eveningNotification = 135 // Declara una constante 'eveningNotification' con
valor 135.
```

```
    printNotificationSummary(morningNotification) // Llama a
printNotificationSummary con la cantidad de notificaciones matutinas.
```

```

    printNotificationSummary(eveningNotification) // Llama a
    printNotificationSummary con la cantidad de notificaciones vespertinas.

}

fun printNotificationSummary(numberOfMessages: Int) { // Define una función que
    toma un parámetro entero para la cantidad de mensajes.

    if (numberOfMessages < 100) { // Verifica si la cantidad de mensajes es menor a
    100.

        println("You have ${numberOfMessages} notifications.") // Imprime la cantidad
        exacta de notificaciones si es menor a 100.

    } else { // Se ejecuta si la cantidad de mensajes es 100 o más.

        println("Your phone is blowing up! You have 99+ notifications.") // Imprime un
        mensaje indicando 99+ notificaciones.

    }

}*/

```

Ejecucion:

```

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

```

Descripción General: Este programa calcula e imprime el precio de entrada al cine para personas de distintas edades, considerando si es lunes o no. Utiliza una función llamada `ticketPrice` que determina el precio según la edad y el día.

```
// Precio de entradas de cine
```

```
/*fun main() { // Define la función principal, el punto de entrada del programa.
```

```
    val child = 5 // Declara una constante 'child' con edad 5.
```

```
    val adult = 28 // Declara una constante 'adult' con edad 28.
```

```
    val senior = 87 // Declara una constante 'senior' con edad 87.
```

```
    val isMonday = true // Declara una constante 'isMonday' establecida en
    verdadero.
```

```

println("The movie ticket price for a person aged $child is \${ticketPrice(child,
isMonday)}") // Imprime el precio del boleto para un niño.

println("The movie ticket price for a person aged $adult is \${ticketPrice(adult,
isMonday)}") // Imprime el precio del boleto para un adulto.

println("The movie ticket price for a person aged $senior is
\${ticketPrice(senior, isMonday)}") // Imprime el precio del boleto para un adulto
mayor.

}

fun ticketPrice(age: Int, isMonday: Boolean): Int { // Define una función que calcula
el precio del boleto según edad y día.

    return when(age) { // Usa una expresión 'when' para evaluar la edad y devolver el
precio correspondiente.

        in 0..12 -> 15 // Devuelve 15 para edades de 0 a 12 (niños).

        in 13..60 -> if (isMonday) 25 else 30 // Devuelve 25 si es lunes, de lo contrario
30, para edades de 13 a 60 (adultos).

        in 61..100 -> 20 // Devuelve 20 para edades de 61 a 100 (adultos mayores).

        else -> -1 // Devuelve -1 para edades inválidas (fuera de 0-100).

    }

}*/

```

Ejecucion:

```

The movie ticket price for a person aged 5 is $15.
The movie ticket price for a person aged 28 is $25.
The movie ticket price for a person aged 87 is $20.

```

Descripción General: Este programa convierte temperaturas entre diferentes unidades (Celsius, Fahrenheit, Kelvin) utilizando una función que acepta una fórmula de conversión como parámetro en forma de lambda. La función printFinalTemperature realiza la conversión e imprime el resultado en un formato legible.

//Conversor de temperatura

```

/*fun main() { // Define la función principal, el punto de entrada del programa.

    printFinalTemperature(27.0, "Celsius", "Fahrenheit") { 9.0 / 5.0 * it + 32 } //
    Convierte 27°C a Fahrenheit e imprime el resultado.

    printFinalTemperature(350.0, "Kelvin", "Celsius") { it - 273.15 } // Convierte 350K
    a Celsius e imprime el resultado.

    printFinalTemperature(10.0, "Fahrenheit", "Kelvin") { 5.0 / 9.0 * (it - 32) + 273.15 }
    // Convierte 10°F a Kelvin e imprime el resultado.

}

fun printFinalTemperature( // Define una función para convertir e imprimir
    temperaturas.

    initialMeasurement: Double, // Parámetro para el valor inicial de la temperatura.

    initialUnit: String, // Parámetro para la unidad inicial (por ejemplo, Celsius).

    finalUnit: String, // Parámetro para la unidad objetivo (por ejemplo, Fahrenheit).

    conversionFormula: (Double) -> Double // Función lambda para realizar la
    conversión.

){

    val finalMeasurement = String.format("%.2f",
    conversionFormula(initialMeasurement)) // Aplica la conversión y formatea a dos
    decimales.

    println("$initialMeasurement degrees $initialUnit is $finalMeasurement degrees
    $finalUnit.") // Imprime el resultado de la conversión.

}*/

```

```

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.

```

Descripción General: Este programa define una clase Song que representa una canción. La clase tiene propiedades como el título, el artista, el año de publicación y el número de reproducciones. Además, se incluye una propiedad calculada isPopular que indica si la canción es popular, basada en el número

de reproducciones. El programa crea una instancia de la clase Song, imprime su descripción y verifica si es popular.

```
//Catalogo de canciones

/*fun main() {

    val brunoSong = Song("We Don't Talk About Bruno", "Encanto Cast", 2022,
    1_000_000) brunoSong.printDescription() println(brunoSong.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.")
}

}*/
```

Ejecucion:

```
We Don't Talk About Bruno, performed by Encanto Cast, was released in 2022.
true
```

Descripción General: Este programa define una clase Person que representa el perfil de una persona en línea, con información como el nombre, la edad, el pasatiempo y un referente (otra persona). La clase tiene una función showProfile() que imprime un perfil detallado, y se crean dos instancias de la clase Person para mostrar cómo funciona el sistema de referencia.

```
//Perfil de internet /*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")

}

}*/
```

Ejecucion:

```
Name: Amanda
Age: 33
Likes to play tennis. Doesn't have a referrer.

Name: Atiqah
Age: 28
Likes to climb. Has a referrer named Amanda, who likes to play tennis.
```

Descripción General: Este programa define una clase Phone y una subclase FoldablePhone que representa un teléfono plegable. La clase Phone tiene métodos para encender y apagar la luz de la pantalla, mientras que la subclase FoldablePhone añade la lógica para verificar si el teléfono está plegado o desplegado antes de permitir que se encienda la pantalla. Se usan propiedades para gestionar el estado de la luz de la pantalla y el estado del plegado del teléfono.

// Telefonos plegables

open class Phone{

var isScreenLightOn: Boolean = false // Propiedad para el estado de la luz de la pantalla, por defecto apagado.

) {

// Define un método abierto para encender la luz de la pantalla.

open fun switchOn() {

// Establece el estado de la luz de la pantalla en verdadero.

isScreenLightOn = true

}

// Define un método para apagar la luz de la pantalla.

fun switchOff() {

// Establece el estado de la luz de la pantalla en falso.

isScreenLightOn = false

```
}
```

```
// Define un método para verificar el estado de la luz de la pantalla.
```

```
fun checkPhoneScreenLight() {
```

```
    // Asigna "on" o "off" según el estado de la luz de la pantalla.
```

```
    val phoneScreenLight = if (isScreenLightOn) "on" else "off"
```

```
    // Imprime el estado de la luz de la pantalla.
```

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

```
}
```

```
}
```

```
class FoldablePhone(
```

```
    var isFolded: Boolean = true // Propiedad para el estado plegado, por defecto plegado.
```

```
) : Phone() {
```

```
    // Sobrescribe el método switchOn para agregar lógica de plegado.
```

```
    override fun switchOn() {
```

```
        // Verifica si el teléfono no está plegado.
```

```
        if (!isFolded) {
```

```
            // Enciende la luz de la pantalla solo si está desplegado.
```

```
            isScreenLightOn = true
```

```
        }
```

```
    }
```

```
    // Define un método para plegar el teléfono.
```

```
    fun fold() {
```

```
// Establece el estado plegado en verdadero.

isFolded = true

}

// Define un método para desplegar el teléfono.

fun unfold() {

    // Establece el estado plegado en falso.

    isFolded = false

}

}

fun main() {

    // Crea un nuevo objeto FoldablePhone, inicialmente plegado.

    val newFoldablePhone = FoldablePhone()

    // Intenta encender la pantalla (no funcionará porque el teléfono está plegado).

    newFoldablePhone.switchOn()

    // Imprime el estado de la luz de la pantalla (esperado: apagado).

    newFoldablePhone.checkPhoneScreenLight()

    // Despliega el teléfono.

    newFoldablePhone.unfold()

    // Enciende la pantalla (funciona porque el teléfono está desplegado).

    newFoldablePhone.switchOn()

    // Imprime el estado de la luz de la pantalla (esperado: encendido).

    newFoldablePhone.checkPhoneScreenLight()

}
```

Ejecucion:

```
The phone screen's light is off.  
The phone screen's light is on.
```