INSTITUTO TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

Tarea No. 5

Control de Versiones GIT y otros ejercicios

Pérez Anastasio Karla Zafiro - 20070574

Garay Hernandez Miguel Enrique - 20070600

Programación Nativa Para Móviles

09:00 - 10:00

Jorge Peralta Escobar

Marzo del 2025

Ciudad Madero, Tamaulipas, México.

ÍNDICE

REPOSITORIO

https://github.com/Zafirows/Programacion-nativa.git

CAPÍTULO 1. EMPEZANDO CON KOTLIN

Hola Mundo

```kotlin
//CAPITULO 1 -Empezando con Kotlin

// Este archivo pertenece al paquete llamado "my.program"

package my.program


// Función principal del programa: el punto de entrada cuando se ejecuta

fun main(args: Array<String>) {

    // Imprime el mensaje "Hello, world!" en la consola

    println("Hello, world!")

}
```

Hola mundo usando una declaración de objeto

```kotlin
//CAPITULO 1 -Empezando con Kotlin

// Este archivo pertenece al paquete llamado "my.program"

package my.program


// Se define un objeto llamado App. En Kotlin, 'object' crea un singleton (una única instancia).

object App {


    // La anotación @JvmStatic indica que esta función debe ser tratada como un método 'static' en Java.

    // Esto es útil si quieres que este método sea llamado desde código Java o por el sistema como punto de entrada.
```

```
    @JvmStatic
    fun main(args: Array<String>) {
        // Imprime el mensaje "Hello World" en la consola
        println("Hello World")
    }
}
```

Hola mundo usando un objeto compañero

```
//CAPITULO 1 -Empezando con Kotlin
// Este archivo pertenece al paquete llamado "my.program"
package my.program


// Se define una clase llamada App
class App {


    // companion object es como una forma de definir miembros estáticos en Kotlin
    companion object {


        // La anotación @JvmStatic hace que la función sea accesible como método estático desde Java.
        // También permite que el sistema reconozca esta función como el punto de entrada (main)
        @JvmStatic
        fun main(args: Array<String>) {
            // Imprime el mensaje "Hello World" en la consola
            println("Hello World")
        }
    }
}
```

Principales métodos utilizando varargs.

```
//CAPITULO 1 -Empezando con Kotlin

// Este archivo pertenece al paquete llamado "my.program"

package my.program


// Función principal del programa con un parámetro 'args' de tipo variable (vararg)

// 'vararg' permite pasar cero o más argumentos de tipo String

fun main(vararg args: String) {

   // Imprime el mensaje "Hello, world!" en la consola

   println("Hello, world!")

}
```

Lectura de entrada desde la línea de comandos

```
fun main(args: Array<String>) {

   // Imprime en pantalla un mensaje para pedir dos números al usuario

   println("Enter Two number")


   // Lee una línea del input del usuario (readLine()), que puede ser nula

   // El operador '!!' fuerza a Kotlin a tratar el resultado como no nulo

   // Si es nulo, lanza una excepción NullPointerException (NPE)

    // Luego divide la línea en dos partes usando el espacio ' ' como separador

    // Finalmente, usa destructuración para asignar las dos partes a las variables 'a' y 'b'

   var (a, b) = readLine()!!.split(' ')


    // Llama a la función maxNum con los dos números convertidos a Int y muestra el resultado

   println("Max number is : ${maxNum(a.toInt(), b.toInt())}")

}


// Función que recibe dos enteros y devuelve el mayor de ellos

fun maxNum(a: Int, b: Int): Int {

   // Variable para almacenar el valor máximo

   // Usa una expresión if para decidir cuál número es mayor

   var max = if (a > b) {
```

```
        // Si 'a' es mayor, imprime su valor y retorna 'a'
        println("The value of a is $a")
        a
    } else {
        // Si 'b' es mayor o igual, imprime su valor y retorna 'b'
        println("The value of b is $b")
        b
    }
    // Retorna el número máximo encontrado
    return max
}
```

CAPÍTULO 2. ADVERTENCIAS EN KOTLIN

Llamando a un toString () en un tipo anulable

```
//CAPITULO 2 -Advertencias con Kotlin

// 'text' es una variable inmutable (val) que obtiene el texto del textField de forma segura
val text = view.textField?.text?.toString() ?: ""
```

CAPÍTULO 3. ANOTACIONES

Meta-anotaciones

```
//CAPITULO 3 -Anotaciones

// ========================================

// Definición de una anotación personalizada

// ========================================


@Target(
    AnnotationTarget.CLASS,             // Se puede usar en clases
    AnnotationTarget.FUNCTION,          // Se puede usar en funciones
    AnnotationTarget.VALUE_PARAMETER,   // Se puede usar en parámetros
```

```
    AnnotationTarget.EXPRESSION         // Se puede usar en expresiones
)
@Retention(AnnotationRetention.SOURCE)    //  Solo  vive  en  el  código
fuente (no en el bytecode)
@MustBeDocumented                                  // Será visible en la
documentación generada
annotation class Fancy                          // Se define la anotación
llamada @Fancy


// ========================================
// Uso de la anotación en diferentes partes
// ========================================


@Fancy // Aplicado a una clase
class MyClass {


    @Fancy // Aplicado a una función
      fun decoratedFunction(@Fancy name: String) { // Aplicado a un
parámetro
          val result = @Fancy "Decorated String"    // Aplicado a una
expresión
        println(result)
    }
}
```

CAPÍTULO 4. ARRAYS

Arreglos Genéricos

```
//CAPITULO 4 -Arrays
fun main() {
    // Crea un array vacío de tipo String
    // No contiene ningún elemento, y su tamaño es 0
```

```
    val empty = emptyArray<String>()

    // Crea un array de tamaño 5
    // Cada elemento se inicializa con el texto "Item #<índice>"
    // Resultado inicial: ["Item #0", "Item #1", "Item #2", "Item #3",
"Item #4"]
    val strings = Array<String>(5) { i -> "Item #$i" }

    // Modifica el valor en la posición 2 (tercer elemento)
    // Ahora el array queda así: ["Item #0", "Item #1", "ChangedItem",
"Item #3", "Item #4"]
    strings[2] = "ChangedItem"

    // Imprime el contenido del array para verificar los cambios
    println(strings.joinToString(prefix = "[", postfix = "]", separator
= ", "))
}
```

Arreglos Primitivos

```
//CAPITULO 4 -Arrays
fun main() {
    // Crea un arreglo de enteros (tipo IntArray) con los valores 1, 2
y 3
    val intArray = intArrayOf(1, 2, 3)

    // Crea un arreglo de números con punto decimal (tipo DoubleArray)
    // con los valores 1.2 y 5.0
    val doubleArray = doubleArrayOf(1.2, 5.0)

    // Imprime el contenido de ambos arreglos en formato legible
    println("intArray: ${intArray.joinToString(", ")}")       //
Salida: intArray: 1, 2, 3
    println("doubleArray: ${doubleArray.joinToString(", ")}") //
Salida: doubleArray: 1.2, 5.0
}
```

Iterar Array

```
//CAPITULO 4 -Arrays
fun main() {
    // Crea un array de tamaño 5
```

```
    // Cada elemento se inicializa con el cuadrado del índice (i * i),
convertido a String
    // Resultado: ["0", "1", "4", "9", "16"]
    val asc = Array(5) { i -> (i * i).toString() }

    // Recorre el array con un bucle for-each
    // En cada iteración imprime el valor del elemento actual
    for (s in asc) {
        println(s)
    }
}
```

CAPÍTULO 5. BUCLES EN KOTLIN

Repetir una acción x veces

```
//CAPITULO 5 -Bucles en kotlin

fun main() {

    // La función repeat(n) ejecuta un bloque de código 'n' veces.

    // En este caso, se repetirá 10 veces.

    repeat(10) { i -> // 'i' es el índice de la iteración (de 0 a 9)

          println("Loop iteration ${i + 1}") // Imprime el número de la
iteración (de 1 a 10)

    }

}
```

Bucle sobre iterables

```
//CAPITULO 5 -Bucles en kotlin

fun main() {

    // Crea una lista inmutable de Strings con tres elementos: "Hello",
"World" y "!"

    val list = listOf("Hello", "World", "!")


    // Recorre cada elemento de la lista usando un bucle for-each
```

```
    for (str in list) {
        print(str) // Imprime cada elemento sin salto de línea
    }
}
```

Iterando sobre un mapa

```
//CAPITULO 5 -Bucles en kotlin
fun main() {
     // Crea un mapa (mapa inmutable) con claves enteras y valores de tipo String
    // El mapa contiene las siguientes asociaciones:
    // 1 -> "foo", 2 -> "bar", 3 -> "baz"
    val map = mapOf(1 to "foo", 2 to "bar", 3 to "baz")

    // Recorre el mapa usando un bucle for-each
    // Desestructura cada entrada en 'key' y 'value'
    for ((key, value) in map) {
        // Imprime cada par clave-valor en formato: Map[clave] = valor
        println("Map[$key] = $value")
    }
}
```

CAPÍTULO 6. COLECCIONES

Usando la lista

```
//CAPITULO 6 -Colecciones
fun main() {
```

```
      // Crea una lista inmutable (no se puede modificar) con tres elementos tipo String
    val list = listOf("Item 1", "Item 2", "Item 3")


    // Imprime la lista completa en una sola línea
    // La salida será en el formato predeterminado de listas en Kotlin
    println(list)
}
```

Usando el mapa

```
//CAPITULO 6 -Colecciones
fun main() {
    // Crea un mapa inmutable (no modificable) con claves de tipo Int y valores de tipo String
    val map = mapOf(
        1 to "Item 1",
        2 to "Item 2",
        3 to "Item 3"
    )


    // Imprime todo el contenido del mapa
      // Kotlin muestra los pares clave-valor en formato: {1=Item 1, 2=Item 2, 3=Item 3}
    println(map)
}
```

CAPÍTULO 7. CONFIGURANDO LA COMPILACIÓN DE KOTLIN

```
// (Ejemplos de configuración Gradle - no aplicable en un archivo .kt)
```

CAPÍTULO 8. COROUTINES

```
//CAPITULO 8 -Coutines

// Importa el paquete necesario para usar corrutinas

/*import kotlinx.coroutines.*


//  Función  principal  que  se  ejecuta  en  un  contexto  bloqueante  de
corrutina

fun main() = runBlocking {

      //  Inicia  una  nueva  corrutina  en  paralelo  dentro  del  bloque
runBlocking

    launch {

         // Suspende la ejecución de esta corrutina durante 1 segundo
(1000 milisegundos)

        delay(1000L)

        // Después de 1 segundo, imprime "World!"

        println("World!")

    }

     // Mientras la corrutina lanzada está en espera, esto se ejecuta
inmediatamente

    println("Hello,")

     // runBlocking esperará automáticamente a que todas las corrutinas
hijas finalicen

}

*/

Nota: Requiere dependencia kotlinx-coroutines-core
```

CAPÍTULO 9. DECLARACIONES CONDICIONALES

Declaración if estándar

```
//CAPITULO 9 -Declaraciones condicionales
fun main() {
```

```
     val str = "Hello!"   // Declara una variable 'str' con el valor
"Hello!"

    // Verifica si la longitud de la cadena es 0 (cadena vacía)
    if (str.length == 0) {
          print("Empty string!")   // Si la cadena está vacía, imprime
este mensaje
    }
    // Si no está vacía, verifica si la longitud es mayor que 5
    else if (str.length > 5) {
           print("Long string!")    // Si es mayor que 5, imprime este
mensaje
    }
    // Si no cumple ninguna de las condiciones anteriores
    else {
          print("Short string!")   // Imprime este mensaje para cadenas
con longitud entre 1 y 5
    }
}
```

When-statement

```
//CAPITULO 10 -When-statement
fun main() {
    val str = "Hello!"

    // Uso de 'when' sin argumento para evaluar condiciones booleanas
    when {
         str.length == 0 -> print("Empty string!")  // Si la cadena está
vacía
           str.length > 5 -> print("Long string!")     // Si la cadena
tiene más de 5 caracteres
         else -> print("Short string!")              // Si no se cumple
ninguna condición anterior
    }
}
```

CAPÍTULO 10. DELEGACIÓN DE CLASE

```
//CAPITULO 10 -Delegación de clase
// Define una interfaz llamada Foo con una función abstracta example()
```

```
interface Foo {
    fun example()
}

// Define una clase Bar que tiene su propia función example()
// Esta función imprime "Hello, world!"
class Bar {
    fun example() {
        println("Hello, world!")
    }
}

// Define una clase Baz que implementa la interfaz Foo
// En lugar de implementar la función example() directamente,
// delega su implementación a un objeto de tipo Bar que recibe en el constructor
class Baz(b: Bar) : Foo by b
```

CAPÍTULO 11. EDIFICIO DSL (PROFUNDIZACIÓN)

```
//CAPITULO 11 -Edificio DSL (Profundización)

// Define una clase llamada Person con dos propiedades: name y age

class Person {

    var name: String = ""

    var age: Int = 0

}



// Define una función de ayuda llamada 'person' que recibe un bloque de código

// Ese bloque tiene como receptor a un objeto de tipo Person

// 'Person.() -> Unit' significa que dentro del bloque puedes acceder a las propiedades de Person directamente

fun person(block: Person.() -> Unit): Person {

    // Crea una nueva instancia de Person y aplica el bloque sobre ella

    return Person().apply(block)

}
```

```
// Llama a la función 'person' y pasa un bloque de configuración
// Dentro del bloque, puedes acceder a 'name' y 'age' como si estuvieras dentro de la clase
val p = person {
    name = "Ana"  // Establece el nombre en "Ana"
    age = 30      // Establece la edad en 30
}
```

CAPÍTULO 12. ENUMERAR

```
//CAPITULO 12 -Enumerar
// Definición de un enum llamado Color
// Cada valor del enum tiene asociado un parámetro 'rgb' de tipo Int
enum class Color(val rgb: Int) {
    RED(0xFF0000),   // Color rojo con valor hexadecimal RGB
    GREEN(0x00FF00), // Color verde con valor hexadecimal RGB
    BLUE(0x0000FF)   // Color azul con valor hexadecimal RGB
}
```

CAPÍTULO 13. EQUIVALENTES DE FLUJO DE JAVA 8

Filtrar una lista

```
//CAPITULO 13 -Equivalentes de flujo de Java 8
// Define una lista inmutable de números del 1 al 6
val numbers = listOf(1, 2, 3, 4, 5, 6)

// Crea una nueva lista llamada 'even' que contiene
// solo los números pares de la lista 'numbers'
// La función 'filter' recibe una lambda que devuelve true
```

```
// para los elementos que queremos mantener

val even = numbers.filter { it % 2 == 0 }
```

Convertir elementos y concatenarlos

```
//CAPITULO 13 -Equivalentes de flujo de Java 8

// Crea una lista inmutable llamada 'things' con los números 1, 2 y 3

val things = listOf(1, 2, 3)


// Une los elementos de la lista en una sola cadena separada por ", "

// La función joinToString convierte la lista a String con separador personalizado

val joined = things.joinToString(", ")
```

CAPÍTULO 14. EXCEPCIONES

```
//CAPITULO 14 -Excepciones

fun example() {

    try {

        // Aquí va el código que puede lanzar una excepción

            // Por ejemplo, acceso a archivos, operaciones que pueden fallar, etc.

    } catch (e: Exception) {

          // Este bloque se ejecuta si ocurre cualquier excepción del tipo Exception

        // 'e.message' contiene el mensaje de error de la excepción

        println("Error: ${e.message}")

    } finally {

        // Este bloque se ejecuta siempre, ocurra o no una excepción

        // Se usa para liberar recursos o hacer limpieza necesaria

        println("Cleanup")

    }
```

```
}
```

CAPÍTULO 15. EXTENSIONES KOTLIN PARA ANDROID (COMPLETO)

```
//CAPITULO 15 -Extensiones Kotlin para Android (Completo)

// Importa extensiones sintéticas para acceder directamente a las vistas del layout activity_main

/*import kotlinx.android.synthetic.main.activity_main.*


class MainActivity : AppCompatActivity() {


    // Método que se llama al crear la actividad

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)


          // Establece el layout activity_main como contenido de esta actividad

        setContentView(R.layout.activity_main)


           // Accede directamente al TextView con id 'textView' y le asigna un texto

        textView.text = "Hola Kotlin"


         // Configura el botón con id 'button' para que al hacer click muestre un mensaje Toast

        button.setOnClickListener {

            Toast.makeText(this, "Click!", Toast.LENGTH_SHORT).show()

        }

    }

}
```

```
*/
```

CAPÍTULO 16. FUNCIONES

Función básica

```
//CAPITULO 16 -Funciones

// Función que recibe un parámetro 'name' de tipo String

// y devuelve un String con un mensaje personalizado

fun sayMyName(name: String): String {

    // Retorna el mensaje "Your name is " seguido del valor de 'name'

    return "Your name is $name"

}
```

Función abreviada

```
//CAPITULO 16 -Funciones

// Función con cuerpo de expresión única que recibe un String 'name'

// y devuelve un String con el mensaje personalizado

fun sayMyNameShort(name: String) = "Your name is $name"
```

Lambda

```
//CAPITULO 16 -Funciones

// Define una función lambda llamada 'isPositive' que recibe un entero
'x'

// y devuelve true si 'x' es mayor que 0, false en caso contrario

val isPositive = { x: Int -> x > 0 }


// Aplica la función 'filter' a la lista de números [-2, -1, 0, 1, 2]

// 'filter' selecciona solo los elementos para los cuales la función
lambda retorna true

val filtered = listOf(-2, -1, 0, 1, 2).filter(isPositive)
```

```
// Resultado: filtered = [1, 2]
```

CAPÍTULO 17. FUNDAMENTOS DE KOTLIN (COMPLETO)

Ejemplo de seguridad nula

```
//CAPITULO 16 -Fundamentos de Kotlin (Completo)

// Definimos una función llamada getLength que recibe un parámetro llamado 'text'.

// Este parámetro puede ser un String o null (por eso se usa String?).

fun getLength(text: String?): Int {


     // Usamos el operador de seguridad null (?.) para acceder a la propiedad 'length' del String.

    // Si 'text' no es null, devuelve su longitud.

    // Si 'text' es null, se usa el operador Elvis (?:) para devolver 0 como valor por defecto.

    return text?.length ?: 0

}
```

Smart casts

```
//CAPITULO 17 -Fundamentos de Kotlin (Completo)

// Definimos una función llamada 'demo' que recibe un parámetro 'x' de tipo Any.

// 'Any' es el tipo base en Kotlin, puede representar cualquier tipo de dato (String, Int, etc.).

fun demo(x: Any) {


     // Usamos una estructura 'if' con el operador 'is' para verificar si 'x' es de tipo String.

    if (x is String) {
```

```
            // Dentro de este bloque, Kotlin automáticamente convierte (castea) 'x' a tipo String.
            // Por eso podemos acceder directamente a propiedades de String, como 'length'.
            println(x.length)
        }
}
```

CAPÍTULO 18. GAMAS

```
//CAPITULO 18 -Gamas
// Primer bucle: recorre los números del 1 al 10 (ambos inclusive)
for (i in 1..10) print(i)
// Salida: 12345678910
// El operador '..' crea un rango del 1 al 10, y 'print(i)' imprime cada número sin salto de línea
// Segundo bucle: recorre los números del 10 al 1, en pasos de 2
for (i in 10 downTo 1 step 2) print(i)
// Salida: 108642
// 'downTo' crea un rango descendente desde 10 hasta 1
// 'step 2' indica que se avanza de 2 en 2 (10, 8, 6, etc.)
```

CAPÍTULO 19. GENÉRICOS

```
//CAPITULO 19 -Genéricos
//  Definimos una clase genérica llamada Box, que acepta un tipo genérico T
class Box<T>(t: T) {
    // Creamos una propiedad mutable llamada 'value', que almacena el valor recibido
    var value = t
```

```
}

// Creamos una instancia de Box con el tipo específico Int y le pasamos el valor 1

val box = Box<Int>(1)
```

CAPÍTULO 20. HERENCIA DE CLASE

```
//CAPITULO 20 -Herencia de clase

// Declaramos una clase abierta llamada Base.

// 'open' permite que esta clase pueda ser heredada por otras (por defecto, las clases en Kotlin son 'final').

open class Base(val prop: String)

// Declaramos una clase llamada Derived que hereda de Base.

// En el constructor de Derived recibimos un parámetro 'prop', que se pasa al constructor de la clase Base.

class Derived(prop: String) : Base(prop)
```

CAPÍTULO 21. INSTRUMENTOS DE CUERDA

```
//CAPITULO 21 -Instrumentos de cuerda

val str = "Hello"          // Creamos una cadena de texto con el valor "Hello"

println(str[1])               // Imprime el carácter en la posición 1 (empezando desde 0). Resultado: 'e'

val multiline = """

    Line 1

    Line 2

""".trimIndent()
```

CAPÍTULO 22. INTERFACES

```
//CAPITULO 22 -Interfaces

// Definimos una interfaz llamada MyInterface
```

```kotlin
interface MyInterface {


       // Declaramos un método abstracto 'bar' que las clases que implementen esta interfaz deben definir
    fun bar()


    // Declaramos un método 'foo' con una implementación por defecto
    fun foo() {
        println("Default implementation")
    }
}


// ===== EJEMPLOS COMBINADOS KOTLIN (Capítulos 23-38) =====
// Notas importantes:
// 1. Algunos ejemplos requieren dependencias específicas (Android, JUnit, etc.)
// 2. Los ejemplos de DSL y builders son simplificados para mostrar el concepto básico
// 3. Las corrutinas requieren el contexto adecuado para ejecutarse
```

CAPÍTULO 23. JUnit

```kotlin
//CAPITULO 23 -JUnit
// Esta clase de prueba usa JUnit, por lo que necesitas agregar la dependencia de JUnit en tu proyecto.


class MyTest {


     // Declaramos una regla de JUnit llamada tempFolder que crea una carpeta temporal para las pruebas.
    // La anotación '@get:Rule' indica que esto es una regla que JUnit ejecutará antes y después de cada test.
```

```
    @get:Rule
    val tempFolder = TemporaryFolder()


    // Definimos una función de prueba anotada con @Test
    @Test
    fun test() {
            // Creamos un nuevo archivo temporal dentro de la carpeta temporal llamada "test.txt"
        val file = tempFolder.newFile("test.txt")


         // Aquí irían las pruebas que involucren el archivo temporal, por ejemplo, escribir o leer datos
    }
}
```

CAPÍTULO 24. KOTLIN PARA DESARROLLADORES DE JAVA

```
//Capítulo 24 -Kotlin para desarrolladores de Java
// Diferencias clave con Java:
val name: String = "Kotlin" // Tipo explícito (opcional)
var mutableVar = 42         // Variable mutable
val immutableVal = "Hola"  // Variable inmutable (final)
```

if como expresión

```
//Capítulo 24 -Kotlin para desarrolladores de Java
// 'max' obtiene el valor máximo entre 'a' y 'b' usando una expresión if
val max = if (a > b) a else b
```

when (mejorado switch)

```
//Capítulo 24 -Kotlin para desarrolladores de Java
```

```
// 'description' recibe un valor según la evaluación de condiciones en el 'when'
val description = when {
     x == 0 -> "cero"         // Si x es igual a 0, description será "cero"
     x > 0 -> "positivo"     // Si x es mayor que 0, description será "positivo"
    else -> "negativo"      // Si ninguna condición anterior se cumple, description será "negativo"
}
```

CAPÍTULO 25. LAMBDAS BÁSICAS

```
//CAPITULO 24 -Lambdas básicas
// Definimos una lambda (función anónima) que recibe dos enteros y devuelve su suma
val sum = { a: Int, b: Int -> a + b }
// Creamos una lista de enteros y usamos la función 'filter' con una lambda para obtener solo los números positivos
val positives = listOf(-2, -1, 0, 1, 2).filter { it > 0 }
```

Lambda con receptor

```
//CAPITULO 24 -Lambdas básicas
// Creamos un StringBuilder y usamos 'apply' para ejecutar varias operaciones sobre él
val stringBuilder = StringBuilder().apply {
    append("Hello")    // Agrega "Hello" al StringBuilder
    append(" ")        // Agrega un espacio
    append("World")    // Agrega "World"
}
```

CAPÍTULO 26. LOGGIN EN KOTLIN

```
//CAPITULO 26 -Logging en Kotlin

// Requiere dependencia kotlin-logging

/*

private val logger = KotlinLogging.logger {}

class MyClass {

    fun doSomething() {

        logger.info("Información importante")

    }

}

*/
```

CAPÍTULO 27. MÉTODOS DE EXTENSIÓN

```
//CAPITULO 27 -Métodos de extensión

// Función de extensión para String: agrega signos de exclamación al
final

fun String.addEnthusiasm(amount: Int = 1) = this + "!".repeat(amount)


// Ejemplo de uso:

println("Hola".addEnthusiasm(3)) // Imprime: Hola!!!


// Función de extensión para Int: verifica si el número es par

fun Int.isEven() = this % 2 == 0


// Ejemplo de uso:

println(4.isEven()) // Imprime: true
```

CAPÍTULO 28. MODIFICADORES DE VISIBILIDAD

```
//CAPITULO 28 -Modificadores de visibilidad

class VisibilityExample {

    public val publicProp = "Público"        // Visible desde cualquier
parte. Este es el valor por defecto si no se especifica visibilidad.


    private val privateProp = "Privado"     // Solo accesible dentro de
esta clase. Ni siquiera las subclases pueden verlo.


    internal val internalProp = "Interno"   // Visible dentro del mismo
**módulo** (por ejemplo, un mismo proyecto o paquete compilado).


    protected val protectedProp = "Protegido" // Solo accesible dentro
de esta clase y sus **subclases** (incluso si están en otros archivos o
paquetes).

}
```

CAPÍTULO 29. MODISMOS

Data class (DTO)

```
//CAPITULO 29 -Modismos

// La palabra clave `data` indica que esta clase está pensada para
"transportar datos".

// Kotlin generará automáticamente varios métodos útiles (equals,
hashCode, toString, copy, etc.).

data class User(

    val name: String, // Propiedad inmutable (val) que guarda el nombre

    val age: Int      // Propiedad inmutable que guarda la edad

)
```

Uso de let para null safety

```
//CAPITULO 29 -Modismos

val nullableString: String? = "Kotlin" // Variable que puede ser null (tipo String nullable)


// Uso de 'let' con el operador seguro ?. para ejecutar un bloque solo si no es null

nullableString?.let {

    println(it.uppercase()) // Solo se ejecuta si nullableString NO es null

}
```

Uso de apply para inicialización

```
//CAPITULO 29 -Modismos

// Crea un nuevo TextView (una vista de texto) usando el contexto actual (por ejemplo, una Activity)

val myTextView = TextView(context).apply {

    // Asigna el texto que se mostrará en el TextView

    text = "Hola"


    // Asigna el tamaño del texto en puntos flotantes (16f equivale a 16sp)

    textSize = 16f

} // 'apply' devuelve el mismo objeto ya configurado, que se guarda en 'myTextView'
```

CAPÍTULO 30. OBJETOS SINGLETON

```
//CAPITULO 30 -Objetos singleton

// Declaración de un objeto singleton llamado 'DatabaseManager'

object DatabaseManager {
```

```
    // Bloque de inicialización: se ejecuta automáticamente la primera vez que se accede al objeto
    init {
        println("Inicializando conexión a BD") // Simula la conexión a la base de datos
    }

    // Función miembro del objeto que simula la ejecución de una consulta SQL
    fun query(sql: String) {
        // Aquí iría el código para ejecutar la consulta en la base de datos
        println("Ejecutando: $sql") // Para mostrar qué consulta se está ejecutando
    }
}

// ---------------- USO ---------------------

// Llamada al método 'query' del objeto singleton
DatabaseManager.query("SELECT * FROM users")

// Al ejecutar esta línea por primera vez, se imprimirá:
// Inicializando conexión a BD
// Ejecutando: SELECT * FROM users
```

CAPÍTULO 31. PARÁMETROS VARARG

```
//CAPITULO 31 -Parámetros Vararg
```

```kotlin
// Define una función llamada 'printAll' que recibe un número variable de argumentos tipo String
fun printAll(vararg messages: String) {
    // Itera sobre cada mensaje recibido y lo imprime
    for (m in messages) println(m)
}

// ---------------- USO ---------------------

// Llama a la función con tres Strings
printAll("Hola", "Mundo", "Kotlin")

// Resultado en consola:
// Hola
// Mundo
// Kotlin
```

Operador spread

```kotlin
//CAPITULO 31 -Parámetros Vararg
val items = arrayOf("uno", "dos", "tres")
// Crea un arreglo (array) de Strings con tres elementos

printAll(*items)
// Llama a la función printAll usando el operador '*'
// El operador '*' (spread operator) "expande" el array como si escribieras: printAll("uno", "dos", "tres")
```

CAPÍTULO 32. PROPIEDADES DELEGADAS

```kotlin
//CAPITULO 32 -Propiedades delegadas
import kotlin.properties.Delegates
```

```
// Lazy initialization
// Declaración de una propiedad llamada 'lazyValue' que se inicializa de forma "perezosa" (lazy)
val lazyValue: String by lazy {
     println("Calculando valor")  // Este código solo se ejecutará la primera vez que se acceda a 'lazyValue'
     "Resultado"                          // El valor que se asignará a 'lazyValue' después de la primera evaluación
}



// Observable property
// Declara una variable observable llamada 'observed' con valor inicial 0
var observed by Delegates.observable(0) { _, old, new ->
    // Este bloque se ejecuta cada vez que 'observed' cambia de valor
     // '_' ignora el parámetro que representa la propiedad (no se usa aquí)
    // 'old' es el valor anterior de la variable
    // 'new' es el nuevo valor asignado

    println("Cambiado de $old a $new") // Imprime el cambio de valor
}
```

CAPÍTULO 33. RECICLERVIEW EN KOTLIN

```
//CAPITULO 33 -RecyclerView en Kotlin
// Ejemplo simplificado (requiere Android)
/*
class MyAdapter(private val items: List<String>) :
    RecyclerView.Adapter<MyAdapter.ViewHolder>() {
```

```
    class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {
        val textView: TextView = view.findViewById(R.id.textView)
    }

     override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
ViewHolder {
        val view = LayoutInflater.from(parent.context)
            .inflate(R.layout.item_layout, parent, false)
        return ViewHolder(view)
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        holder.textView.text = items[position]
    }

    override fun getItemCount() = items.size
}
*/
```

CAPÍTULO 34. REFLEXION

```
//CAPITULO 34 -Reflexión
// Obtener la referencia a la clase 'String' usando reflection
val stringClass = String::class

// Imprime el nombre simple de la clase (sin paquete)
println(stringClass.simpleName) // Salida: String
// ------------------
// Definición de una data class llamada 'Person' con dos propiedades
```

```
data class Person(val name: String, val age: Int)


// Obtiene todas las propiedades (valores miembros) de la clase 'Person' usando reflection
val properties = Person::class.memberProperties


// Itera sobre cada propiedad y la imprime (por ejemplo: "name" y "age")
properties.forEach { println(it.name) }
```

CAPÍTULO 35. REGEX

```
//CAPITULO 35 -Regex
// Crea una expresión regular que coincide con uno o más dígitos (\d+)
val regex = Regex("\\d+")


// Verifica si toda la cadena "123" coincide con la expresión regular (true)
println(regex.matches("123")) // true


// Verifica si toda la cadena "abc" coincide con la expresión regular (false)
println(regex.matches("abc")) // false




// ------------------


// Expresión regular para capturar fechas en formato dd/mm/yyyy
val dateRegex = Regex("(\\d{2})/(\\d{2})/(\\d{4})")


// Busca la primera coincidencia en la cadena "15/07/2023"
```

```
val matchResult = dateRegex.find("15/07/2023")


// Si hay coincidencia, extrae los grupos capturados y los imprime
matchResult?.let {
    // Grupo 1: día (dos dígitos)
    println("Día: ${it.groupValues[1]}")

    // Grupo 2: mes (dos dígitos)
    println("Mes: ${it.groupValues[2]}")

    // Grupo 3: año (cuatro dígitos)
    println("Año: ${it.groupValues[3]}")
}
```

CAPÍTULO 36. SEGURIDAD NULA

```
//CAPITULO 36 -Seguridad nula
// Safe call operator: accede a 'length' solo si 'nullableString' NO es null, sino devuelve null
val length: Int? = nullableString?.length


// Elvis operator: si 'nullableString?.length' es null, entonces usa 0 como valor por defecto
val safeLength = nullableString?.length ?: 0


// Not-null assertion: fuerza a que 'nullableString' NO sea null; lanza excepción si es null
val forcedLength = nullableString!!.length
```

CAPÍTULO 37. TIPO DE ALIAS

```
//CAPITULO 37 -Tipo de alias

// 'typealias' crea un alias para un tipo complejo, facilitando su uso y lectura


// Alias para una lista de objetos User

typealias UserList = List<User>


// Alias para una función que recibe un User y un String, y devuelve un Boolean

typealias AuthCallback = (User, String) -> Boolean


// Función que recibe una lista de usuarios (UserList) y un callback de autenticación (AuthCallback)

fun processUsers(users: UserList, callback: AuthCallback) {

    // Aquí iría la implementación para procesar usuarios y aplicar el callback

}
```

CAPÍTULO 38. TIPO DE CONSTRUCTORES SEGUROS

```
//CAPITULO 38 -Tipo de constructores seguros

// Clase principal que representa un documento HTML

class HTML {

       // Función para crear un 'body' y aplicar una función de inicialización (DSL)

    fun body(init: Body.() -> Unit) = Body().apply(init)

}


// Clase que representa el body del HTML
```

```
class Body {
      // Función para crear un párrafo <p> y aplicar una función de inicialización
    fun p(init: P.() -> Unit) = P().apply(init).toString()
}

// Clase que representa un párrafo <p>
class P {
    var text = ""  // Contenido del párrafo

    // Convierte el objeto P a su representación HTML en String
    override fun toString() = "<p>$text</p>"
}

// Función para construir un objeto HTML usando un DSL (Domain Specific Language)
fun html(init: HTML.() -> Unit): HTML = HTML().apply(init)



// ---------------------
// Uso del builder:
// Construye un HTML con un body que contiene un párrafo con texto "Hola mundo"
val htmlContent = html {
    body {
        p {
            text = "Hola mundo"
        }
    }
}
```