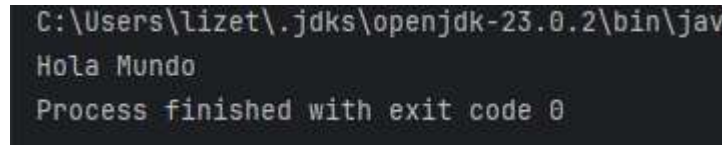


### Proyecto 1:

// Definición de la función principal que actúa como punto de entrada al programa

```
fun main(parametro: Array<String>) {  
    // Imprime "Hola Mundo" en la consola  
    print("Hola Mundo")  
}
```



```
C:\Users\lizet\.jdk\openjdk-23.0.2\bin\java  
Hola Mundo  
Process finished with exit code 0
```

**Documentación:** Este programa básico en Kotlin utiliza la función principal main como punto de entrada, recibiendo un parámetro Array que representa un arreglo de cadenas de texto. Aunque el parámetro no se utiliza en el código, el programa imprime en la consola el mensaje "Hola Mundo", mostrando un ejemplo simple de cómo funciona la impresión en Kotlin.

### Proyecto 2:

// Definición de la función principal que actúa como punto de entrada al programa

```
fun main(parametro: Array<String>) {  
    // Declaración de dos valores constantes: valor1 y valor2  
  
    val valor1 = 100  
    val valor2 = 400  
  
    // Variable mutable donde se almacena la suma de valor1 y valor2  
    var resultado = valor1 + valor2  
  
    // Imprime la suma de los valores en la consola  
    println("La suma de $valor1 + $valor2 es $resultado")  
  
    // Actualización de la variable resultado para almacenar el producto de los valores  
    resultado = valor1 * valor2  
  
    // Imprime el producto de los valores en la consola  
    println("El producto de $valor1 * $valor2 es $resultado")  
}
```

```
C:\Users\lizet\.jdk\openjdk-23.0.2\bin
La suma de 100 + 400 es 500
El producto de 100 * 400 es 40000

Process finished with exit code 0
```

**Documentación:** Este programa en Kotlin utiliza la función principal main como punto de entrada. Declara dos valores constantes, valor1 y valor2, y calcula la suma y el producto de ambos, almacenando los resultados en una variable mutable llamada resultado. Luego, imprime los cálculos en la consola utilizando interpolación de cadenas para mostrar los valores y los resultados correspondientes.

### Proyecto 3:

// Definición de la función principal que actúa como punto de entrada al programa

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

```
    // Solicita al usuario que ingrese el primer valor
```

```
    println("Ingrese primer valor:")
```

```
    // Lee la entrada del usuario y la convierte en un entero
```

```
    val valor1 = readLine()!!.toInt()
```

```
    // Solicita al usuario que ingrese el segundo valor
```

```
    println("Ingrese segundo valor")
```

```
    // Lee la entrada del usuario y la convierte en un entero
```

```
    val valor2 = readLine()!!.toInt()
```

```
    // Calcula la suma de los valores ingresados
```

```
    val suma = valor1 + valor2
```

```
    // Imprime el resultado de la suma en la consola
```

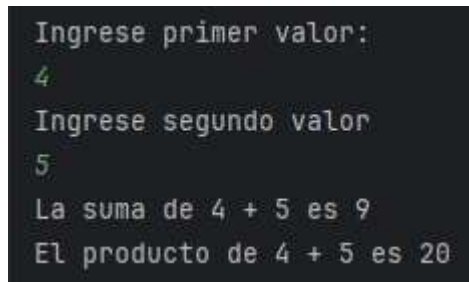
```
    println("La suma de $valor1 + $valor2 es $suma")
```

```
    // Calcula el producto de los valores ingresados
```

```
    val producto = valor1 * valor2
```

```
    // Imprime el resultado del producto en la consola
```

```
println("El producto de $valor1 + $valor2 es $producto")
}
```



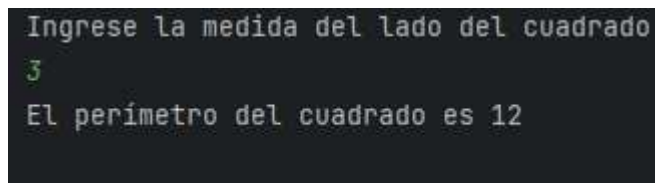
```
Ingrese primer valor:
4
Ingrese segundo valor
5
La suma de 4 + 5 es 9
El producto de 4 + 5 es 20
```

**Documentación:** Este programa en Kotlin utiliza la función principal main como punto de entrada y permite al usuario ingresar dos números desde la consola. Los números se convierten de cadenas a enteros usando `readLine()` y `toInt()`. Luego, el programa calcula la suma y el producto de los valores ingresados y los muestra en la consola utilizando interpolación de cadenas para exhibir los cálculos de manera clara y dinámica.

#### Proyecto 4:

// Definición de la función principal que actúa como punto de entrada al programa

```
fun main(parametro: Array<String>) {
    // Solicita al usuario ingresar la medida de un lado del cuadrado
    println("Ingrese la medida del lado del cuadrado")
    // Lee la entrada del usuario y la convierte en un entero
    val lado = readLine()!!.toInt()
    // Calcula el perímetro del cuadrado multiplicando el lado por 4
    val perimetro = lado * 4
    // Imprime el resultado del perímetro en la consola
    println("El perímetro del cuadrado es $perimetro")
}
```



```
Ingrese la medida del lado del cuadrado
3
El perímetro del cuadrado es 12
```

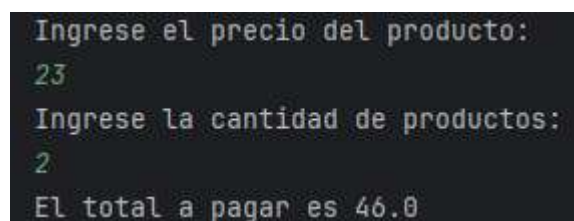
**Documentación:** Este programa en Kotlin permite al usuario calcular el perímetro de un cuadrado ingresando la medida de uno de sus lados desde la consola. Utiliza la función `readLine()` para recibir la entrada del usuario, la convierte a un entero con `toInt()`, y luego calcula el perímetro multiplicando

la longitud del lado por cuatro. Finalmente, muestra el resultado en la consola utilizando interpolación de cadenas para presentar el cálculo.

#### Proyecto 5:

// Definición de la función principal que actúa como punto de entrada al programa

```
fun main(parametro: Array<String>) {  
    // Solicita al usuario ingresar el precio del producto  
    println("Ingresa el precio del producto:")  
  
    // Lee la entrada del usuario y la convierte a un valor de tipo Double  
    val precio = readLine()!!.toDouble()  
  
    // Solicita al usuario ingresar la cantidad de productos  
    println("Ingresa la cantidad de productos:")  
  
    // Lee la entrada del usuario y la convierte a un valor de tipo Int  
    val cantidad = readLine()!!.toInt()  
  
    // Calcula el total multiplicando el precio por la cantidad  
    val total = precio * cantidad  
  
    // Imprime el total calculado en la consola  
    println("El total a pagar es $total")  
}
```



```
Ingresa el precio del producto:  
23  
Ingresa la cantidad de productos:  
2  
El total a pagar es 46.0
```

**Documentación:** Este programa en Kotlin calcula el costo total de una compra, solicitando al usuario que ingrese el precio de un producto y la cantidad de productos adquiridos. Utiliza la función `readLine()` para capturar los valores desde la consola, convierte el precio a tipo `Double` y la cantidad a tipo `Int`, realiza el cálculo del total mediante la multiplicación, y muestra el resultado al usuario con interpolación de cadenas.

#### Proyecto 6:

```
// Definición de la función principal que actúa como punto de entrada al programa
fun main(parametros: Array<String>) {
    // Solicita al usuario que ingrese el sueldo del empleado
    println("Ingrese el sueldo del empleado:")

    // Lee la entrada del usuario y la convierte a un valor de tipo Double
    val sueldo = readLine()!!.toDouble()

    // Condición que verifica si el sueldo es mayor a 3000
    if (sueldo > 3000) {
        // Imprime un mensaje indicando que el empleado debe pagar impuestos
        println("Debe pagar impuestos")
    }
}
```



```
Ingrese el sueldo del empleado:
230
```

**Documentación:** Este programa en Kotlin verifica si un empleado debe pagar impuestos en función de su sueldo. Utiliza la función `readLine()` para capturar el sueldo ingresado por el usuario desde la consola y lo convierte a tipo `Double`. Luego, utiliza una estructura condicional `if` para evaluar si el sueldo supera los 3000, en cuyo caso muestra un mensaje indicando que el empleado debe pagar impuestos.

### Proyecto 7:

```
// Definición de la función principal que actúa como punto de entrada al programa
fun main(parametro: Array<String>) {
    // Solicita al usuario ingresar el primer valor
    println("Ingrese primer valor:")

    // Lee la entrada del usuario y la convierte en un entero
    val valor1 = readLine()!!.toInt()

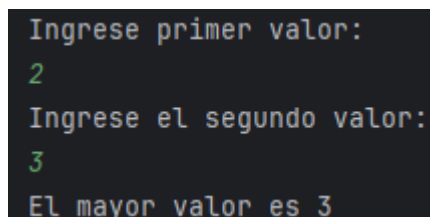
    // Solicita al usuario ingresar el segundo valor
    println("Ingrese el segundo valor:")
```

```

// Lee la entrada del usuario y la convierte en un entero
val valor2 = readLine()!!.toInt()

// Condición para verificar cuál de los dos valores es mayor
if (valor1 > valor2)
    // Si valor1 es mayor, lo imprime en la consola
    println("El mayor valor es $valor1")
else
    // Si valor2 es mayor o igual, lo imprime en la consola
    println("El mayor valor es $valor2")
}

```



```

Ingrese primer valor:
2
Ingrese el segundo valor:
3
El mayor valor es 3

```

**Documentación:** Este programa en Kotlin permite al usuario ingresar dos valores numéricos y determina cuál de los dos es mayor. Utiliza la función `readLine()` para capturar las entradas desde la consola y las convierte a enteros con `toInt()`. Luego, emplea una estructura condicional `if-else` para comparar ambos valores e imprime en la consola el mayor de ellos utilizando interpolación de cadenas.

### Proyecto 8:

```

// Definición de la función principal que actúa como punto de entrada al programa
fun main(parametro: Array<String>) {
    // Solicita al usuario ingresar el primer valor
    println("Ingrese el primer valor:")

    // Lee la entrada del usuario y la convierte en un entero
    val valor1 = readLine()!!.toInt()

    // Solicita al usuario ingresar el segundo valor
    println("Ingrese el segundo valor:")
}

```

```

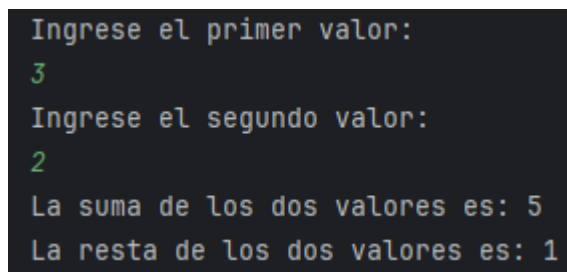
// Lee la entrada del usuario y la convierte en un entero
val valor2 = readLine()!!.toInt()

// Condición para verificar si el primer valor es mayor al segundo
if (valor1 > valor2) {
    // Si valor1 es mayor, se calculan la suma y la resta de los dos valores
    val suma = valor1 + valor2
    val resta = valor1 - valor2

    // Se imprimen los resultados de la suma y la resta en la consola
    println("La suma de los dos valores es: $suma")
    println("La resta de los dos valores es: $resta")
} else {
    // Si valor2 es mayor o igual, se calculan el producto y la división de los dos valores
    val producto = valor1 * valor2
    val division = valor1 / valor2

    // Se imprimen los resultados del producto y la división en la consola
    println("El producto de los dos valores es: $producto")
    println("La división de los dos valores es: $division")
}
}

```



```

Ingrese el primer valor:
3
Ingrese el segundo valor:
2
La suma de los dos valores es: 5
La resta de los dos valores es: 1

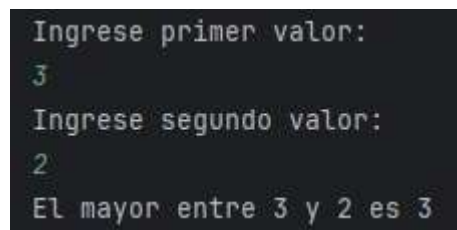
```

**Documentación:** Este programa en Kotlin permite al usuario ingresar dos valores enteros y realiza diferentes operaciones según cuál de los valores sea mayor. Si el primer valor es mayor, calcula la suma y la resta de los dos números; en caso contrario, calcula el producto y la división. Utiliza la función `readLine()` para capturar las entradas, las convierte a enteros con `toInt()` y muestra los resultados de las operaciones en la consola con interpolación de cadenas.

### Proyecto 9:

// Definición de la función principal que actúa como punto de entrada al programa

```
fun main(parametro: Array<String>) {  
    // Solicita al usuario ingresar el primer valor  
    println("Ingrese primer valor:")  
  
    // Lee la entrada del usuario y la convierte en un entero  
    val valor1 = readLine()!!.toInt()  
  
    // Solicita al usuario ingresar el segundo valor  
    println("Ingrese segundo valor:")  
  
    // Lee la entrada del usuario y la convierte en un entero  
    val valor2 = readLine()!!.toInt()  
  
    // Usando una expresión condicional, asigna el valor mayor a la variable `mayor`  
    val mayor = if (valor1 > valor2) valor1 else valor2  
  
    // Imprime el mayor valor en la consola  
    println("El mayor entre $valor1 y $valor2 es $mayor")  
}
```



```
Ingrese primer valor:  
3  
Ingrese segundo valor:  
2  
El mayor entre 3 y 2 es 3
```

**Documentación:** Este programa en Kotlin permite al usuario ingresar dos valores enteros y determina cuál es el mayor entre ellos. Utiliza la función `readLine()` para capturar las entradas de la consola y las convierte a enteros con `toInt()`. Emplea una expresión condicional `if-else` en una sola línea para comparar los valores, asignando el mayor a la variable `mayor`, y muestra el resultado en la consola utilizando interpolación de cadenas.

### Proyecto 10:

// Definición de la función principal que actúa como punto de entrada al programa

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



```

// Solicita al usuario que ingrese un valor entero

print("Ingrese un valor entero:")

// Lee la entrada del usuario y la convierte en un entero
val valor = readLine()!!.toInt()


// Usando una expresión condicional, verifica si el valor es par o impar
val resultado = if (valor % 2 == 0) {

    // Si el número es par, imprime "Cuadrado:" y calcula el cuadrado del valor
    println("Cuadrado:")

    valor * valor

} else {

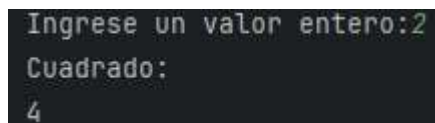
    // Si el número es impar, imprime "Cubo:" y calcula el cubo del valor
    println("Cubo:")

    valor * valor * valor

}

// Imprime el resultado del cálculo en la consola
println(resultado)
}

```



```

Ingrese un valor entero:2
Cuadrado:
4

```

**Documentación:** Este programa en Kotlin permite al usuario ingresar un número entero y determina si es par o impar utilizando el operador módulo (%). Si el número es par, calcula su cuadrado; si es impar, calcula su cubo. El programa presenta el tipo de operación realizada ("Cuadrado" o "Cubo") seguido del resultado, todo mostrado en la consola mediante interpolación de cadenas y estructuras condicionales.

### Proyecto 11:

```

// Definición de la función principal que actúa como punto de entrada al programa
fun main(parametros: Array<String>) {

    // Solicita al usuario ingresar la primera nota
    println("Ingrese primer nota:")
}

```

```
// Lee la entrada del usuario y la convierte en un entero
val nota1 = readLine()!!.toInt()

// Solicita al usuario ingresar la segunda nota
println("Ingrese Segunda nota:")

// Lee la entrada del usuario y la convierte en un entero
val nota2 = readLine()!!.toInt()

// Solicita al usuario ingresar la tercera nota
println("Ingrese Tercera nota:")

// Lee la entrada del usuario y la convierte en un entero
val nota3 = readLine()!!.toInt()

// Calcula el promedio de las tres notas
val promedio = (nota1 + nota2 + nota3) / 3

// Condicional para evaluar el promedio y determinar el estado del estudiante
if (promedio >= 7)
    // Si el promedio es mayor o igual a 7, imprime "Promocionado"
    println("Promocionado")
else
    // Si el promedio es menor o igual a 4, imprime "Regular"
    if (promedio <= 4)
        println("Regular")
    else
        // En cualquier otro caso, imprime "Libre"
        println("Libre")
}
```

```
Ingrese primer nota:
8
Ingrese Segunda nota:
8
Ingrese Tercera nota:
8
Promocionado
```

**Documentación:** Este programa en Kotlin permite al usuario ingresar tres notas numéricas, calcula el promedio de estas y determina el estado del estudiante según la escala definida: "Promocionado" si el promedio es mayor o igual a 7, "Regular" si es menor o igual a 4, y "Libre" para los demás casos. Las entradas son obtenidas desde la consola y convertidas a enteros utilizando `readLine()` y `toInt()`.

### Proyecto 12:

// Definición de la función principal que actúa como punto de entrada al programa

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

```
    // Solicita al usuario ingresar el primer valor
```

```
    print("Ingrese el primer valor:")
```

```
    // Lee la entrada del usuario y la convierte en un entero
```

```
    val valor1 = readLine()!!.toInt()
```

```
    // Solicita al usuario ingresar el segundo valor
```

```
    print("Ingrese el segundo valor:")
```

```
    // Lee la entrada del usuario y la convierte en un entero
```

```
    val valor2 = readLine()!!.toInt()
```

```
    // Condición para verificar si el primer valor es menor que el segundo
```

```
    if (valor1 < valor2) {
```

```
        // Si valor1 es menor, se calculan la suma y la resta de los valores
```

```
        val suma = valor1 + valor2
```

```
        val resta = valor1 - valor2
```

```
        // Se imprimen los resultados de la suma y la resta en la consola
```

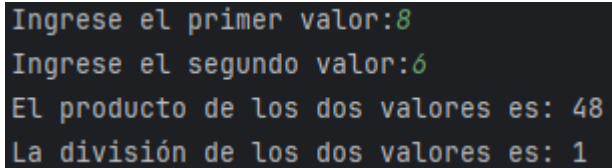
```
        println("La suma de los dos valores es: $suma")
```

```

        println("La resta de los dos valores es: $resta")
    } else {
        // Si valor2 es menor o igual, se calculan el producto y la división de los valores
        val producto = valor1 * valor2
        val division = valor1 / valor2

        // Se imprimen los resultados del producto y la división en la consola
        println("El producto de los dos valores es: $producto")
        println("La división de los dos valores es: $division")
    }
}

```



```

Ingrese el primer valor:8
Ingrese el segundo valor:6
El producto de los dos valores es: 48
La división de los dos valores es: 1

```

**Documentación:** Este programa en Kotlin permite al usuario ingresar dos valores enteros y realiza diferentes operaciones según cuál de los valores sea menor. Si el primer valor es menor que el segundo, calcula la suma y la resta; de lo contrario, calcula el producto y la división. Utiliza la función `readLine()` para capturar las entradas desde la consola, las convierte a enteros con `toInt()`, y muestra los resultados de las operaciones en la consola mediante interpolación de cadenas.

### Proyecto 13:

// Definición de la función principal que actúa como punto de entrada al programa

```

fun main(parametro: Array<String>) {
    // Solicita al usuario ingresar la primera nota
    print("Ingrese primer nota:")

    // Lee la entrada del usuario y la convierte en un entero
    val nota1 = readln().toInt()

    // Solicita al usuario ingresar la segunda nota
    print("Ingrese segunda nota:")

    // Lee la entrada del usuario y la convierte en un entero
    val nota2 = readln().toInt()
}

```

```

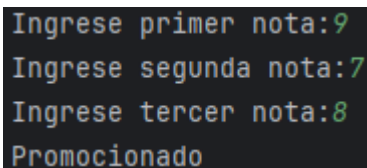
// Solicita al usuario ingresar la tercera nota
print("Ingrese tercer nota:")

// Lee la entrada del usuario y la convierte en un entero
val nota3 = readln().toInt()

// Calcula el promedio de las tres notas
var promedio = (nota1 + nota2 + nota3) / 3

// Condicional para verificar si el promedio es mayor o igual a 7
if (promedio >= 7)
    // Imprime "Promocionado" si la condición es verdadera
    println("Promocionado")
}

```



```

Ingrese primer nota:9
Ingrese segunda nota:7
Ingrese tercer nota:8
Promocionado

```

**Documentación:** Este programa en Kotlin permite al usuario ingresar tres notas numéricas desde la consola, calcula el promedio de las mismas y determina si el estudiante está "Promocionado". Utiliza la función `readln()` para capturar las entradas del usuario, las convierte a enteros con `toInt()`, realiza el cálculo del promedio y evalúa si este es mayor o igual a 7 para imprimir el estado correspondiente.

#### Proyecto 14:

```

// Definición de la función principal que actúa como punto de entrada al programa
fun main(parametro: Array<String>) {

    // Solicita al usuario ingresar un valor entre 1 y 99
    print("Ingrese un valor comprendido entre 1 y 99:")

    // Lee la entrada del usuario y la convierte en un entero
    val num = readln().toInt()

    // Condicional para evaluar si el número es menor que 10

```

```

if (num < 10)

    // Si el número es menor que 10, imprime que tiene un dígito

    println("Tiene un dígito")

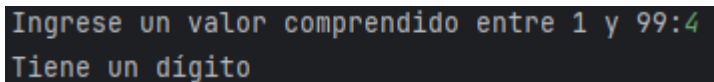
else

    // Si el número no es menor que 10, imprime que tiene dos dígitos

    println("Tiene dos dígitos")

}

```



```

Ingrese un valor comprendido entre 1 y 99:4
Tiene un dígito

```

**Documentación:** Este programa en Kotlin solicita al usuario ingresar un número entero comprendido entre 1 y 99 y determina si el número tiene uno o dos dígitos. Utiliza la función `readln()` para capturar la entrada del usuario y la convierte a entero con `toInt()`. A través de una estructura condicional `if-else`, evalúa si el número es menor que 10 y, dependiendo del resultado, imprime un mensaje indicando si tiene uno o dos dígitos.

#### Proyecto 15:

// Definición de la función principal que actúa como punto de entrada al programa

```

fun main(parametro: Array<String>) {

    // Solicita al usuario ingresar el primer valor

    print("Ingrese primer valor:")

    // Lee la entrada del usuario y la convierte en un entero

    val valor1 = readLine()!!.toInt()


    // Solicita al usuario ingresar el segundo valor

    print("Ingrese segundo valor:")

    // Lee la entrada del usuario y la convierte en un entero

    val valor2 = readLine()!!.toInt()

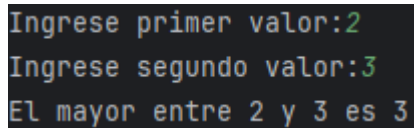

    // Usando una expresión condicional, asigna el mayor valor a la variable `mayor`

    val mayor = if (valor1 > valor2) valor1 else valor2

    // Imprime el mayor valor en la consola

```

```
println("El mayor entre $valor1 y $valor2 es $mayor")
}
```



```
Ingrese primer valor:2
Ingrese segundo valor:3
El mayor entre 2 y 3 es 3
```

**Documentación:** Este programa en Kotlin permite al usuario ingresar dos valores enteros desde la consola y determina cuál de ellos es mayor. Utiliza la función `readLine()` para capturar las entradas del usuario y las convierte a enteros con `toInt()`. Mediante una expresión condicional `if-else` en una sola línea, asigna el mayor valor a la variable `mayor` y presenta el resultado en la consola con interpolación de cadenas.

#### Proyecto 16:

// Definición de la función principal que actúa como punto de entrada al programa

```
fun main(parametro: Array<String>) {
    // Solicita al usuario que ingrese un valor entero
    print("Ingrese un valor entero:")

    // Lee la entrada del usuario y la convierte en un entero
    val valor = readLine()!!.toInt()

    // Usando una expresión condicional, verifica si el valor es par o impar
    val resultado = if (valor % 2 == 0) {
        // Si el número es par, imprime "Cuadrado:" y calcula el cuadrado del valor
        print("Cuadrado:")
        valor * valor
    } else {
        // Si el número es impar, imprime "Cubo:" y calcula el cubo del valor
        print("Cubo:")
        valor * valor * valor
    }

    // Imprime el resultado del cálculo en la consola
    print(resultado)
}
```

```
Ingrese un valor entero:3  
Cubo:27
```

**Documentación:** Este programa en Kotlin solicita al usuario ingresar un número entero y determina si es par o impar utilizando el operador módulo (%). Si el número es par, calcula y muestra su cuadrado; si es impar, calcula y muestra su cubo. Utiliza la función `readLine()` para capturar la entrada del usuario y emplea una expresión condicional `if-else` para realizar los cálculos correspondientes y mostrar el resultado directamente en la consola.

#### Proyecto 17:

// Definición de la función principal que actúa como punto de entrada al programa

```
fun main(parametro: Array<String>) {  
    // Solicita al usuario que ingrese un valor entre 1 y 99  
    print("Ingrese un valor entero comprendido entre 1 y 99:")  
    // Lee la entrada del usuario y la convierte en un entero  
    val valor = readln().toInt()  
  
    // Usando una expresión condicional, determina la cantidad de dígitos  
    var cantidad = if (valor < 10) 1 else 2  
    // Imprime el resultado mostrando el número ingresado y la cantidad de dígitos  
    println("El número $valor tiene $cantidad dígito/s")  
}
```

```
Ingrese un valor entero comprendido entre 1 y 99:34  
El número 34 tiene 2 dígito/s
```

**Documentación:** Este programa en Kotlin solicita al usuario ingresar un número entero comprendido entre 1 y 99, determina si el número tiene uno o dos dígitos utilizando una expresión condicional `if-else`, y presenta el resultado en la consola con interpolación de cadenas, mostrando el número ingresado y la cantidad de dígitos correspondientes.

#### Proyecto 18:

// Definición de la función principal que actúa como punto de entrada al programa

```
fun main(parametros: Array<String>) {  
    // Solicita al usuario ingresar la primera nota  
    print("Ingrese primer nota:")  
    // Lee la entrada del usuario y la convierte en un entero
```



```
val nota1 = readLine()!!.toInt()

// Solicita al usuario ingresar la segunda nota
print("Ingrese segunda nota:")

// Lee la entrada del usuario y la convierte en un entero
val nota2 = readLine()!!.toInt()

// Solicita al usuario ingresar la tercera nota
print("Ingrese tercer nota:")

// Lee la entrada del usuario y la convierte en un entero
val nota3 = readLine()!!.toInt()

// Calcula el promedio de las tres notas
val promedio = (nota1 + nota2 + nota3) / 3

// Condicional para evaluar el promedio y determinar el estado del estudiante
if (promedio >= 7)
    // Imprime "Promocionado" si el promedio es mayor o igual a 7
    print("Promocionado")
else
    // Si el promedio está entre 4 y 6 inclusive, imprime "Regular"
    if (promedio >= 4)
        print("Regular")
    else
        // Si el promedio es menor que 4, imprime "Reprobado"
        print("Reprobado")
}
```

```
Ingrese primer nota:8
Ingrese segunda nota:9
Ingrese tercer nota:0
Regular
```

**Documentación:** Este programa en Kotlin permite al usuario ingresar tres notas numéricas desde la consola, calcula el promedio de las mismas y evalúa el estado del estudiante según la escala definida: "Promocionado" si el promedio es mayor o igual a 7, "Regular" si el promedio está entre 4 y 6 inclusive, y "Reprobado" si el promedio es menor que 4. Utiliza la función `readLine()` para capturar las entradas del usuario y las convierte a enteros con `toInt()`.

#### Proyecto 19:

// Definición de la función principal que actúa como punto de entrada al programa

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

```
    // Solicita al usuario ingresar el primer valor
```

```
    print("Ingrese primer valor:")
```

```
    // Lee la entrada del usuario y la convierte en un entero
```

```
    val valor1 = readln().toInt()
```

```
    // Solicita al usuario ingresar el segundo valor
```

```
    print("Ingrese segundo valor:")
```

```
    // Lee la entrada del usuario y la convierte en un entero
```

```
    val valor2 = readln().toInt()
```

```
    // Solicita al usuario ingresar el tercer valor
```

```
    print("Ingrese tercer valor:")
```

```
    // Lee la entrada del usuario y la convierte en un entero
```

```
    val valor3 = readln().toInt()
```

```
    // Condicional para verificar cuál es el mayor entre los tres valores
```

```
    if (valor1 > valor2)
```

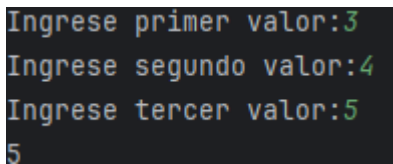
```
        // Si valor1 es mayor que valor2, se compara con valor3
```

```
        if (valor1 > valor3)
```

```

        // Si valor1 es mayor que valor3, se imprime valor1
        print(valor1)
    else
        // Si valor3 es mayor, se imprime valor3
        print(valor3)
    else
        // Si valor2 es mayor o igual que valor1, se compara con valor3
        if (valor2 > valor3)
            // Si valor2 es mayor que valor3, se imprime valor2
            print(valor2)
        else
            // Si valor3 es mayor, se imprime valor3
            print(valor3)
    }

```



```

Ingrese primer valor:3
Ingrese segundo valor:4
Ingrese tercer valor:5
5

```

**Documentación:** Este programa en Kotlin solicita al usuario ingresar tres valores enteros desde la consola y determina cuál de ellos es el mayor. Utiliza la función `readln()` para capturar las entradas del usuario y las convierte a enteros con `toInt()`. Luego, emplea una estructura condicional `if-else` para comparar los tres valores y muestra el mayor número en la consola mediante la función `print()`.

#### Proyecto 20:

// Definición de la función principal que actúa como punto de entrada al programa

```

fun main(parametro: Array<String>) {
    // Solicita al usuario que ingrese un valor entero
    print("Ingrese un valor entero:")

    // Lee la entrada del usuario y la convierte en un entero
    val valor = readln().toInt()

    // Condicional para evaluar si el valor ingresado es cero

```

```

if (valor == 0)

    // Imprime un mensaje indicando que se ingresó el número cero
    println("Se ingresó el cero")

else

    // Condicional para evaluar si el valor ingresado es positivo
    if (valor > 0)

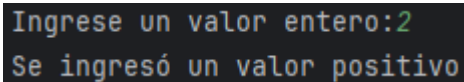
        // Imprime un mensaje indicando que se ingresó un valor positivo
        println("Se ingresó un valor positivo")

    else

        // Si no es cero ni positivo, imprime que se ingresó un valor negativo
        println("Se ingresó un valor negativo")

}

```



```

Ingrese un valor entero:2
Se ingresó un valor positivo

```

**Documentación:** Este programa en Kotlin solicita al usuario ingresar un número entero y, mediante una estructura condicional if-else, evalúa si el valor ingresado es cero, positivo o negativo, mostrando un mensaje correspondiente en la consola. Utiliza la función `readln()` para leer la entrada del usuario y la convierte a entero con `toInt()` antes de realizar las evaluaciones.

### Proyecto 21:

// Definición de la función principal que actúa como punto de entrada al programa

```

fun main(parametro: Array<String>) {

    // Solicita al usuario que ingrese un valor entero con 1, 2 o 3 cifras
    print("Ingrese un valor entero con 1,2 o 3 cifras:")

    // Lee la entrada del usuario y la convierte en un entero
    val valor = readln().toInt()

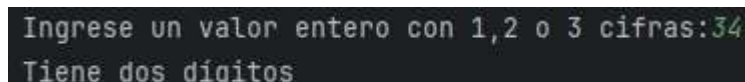
    // Condicional para evaluar si el número tiene una cifra
    if (valor < 10)

```

```

        // Imprime que el número tiene un dígito
        println("Tiene un dígito")
    else
        // Condicional para evaluar si el número tiene dos cifras
        if (valor < 100)
            // Imprime que el número tiene dos dígitos
            println("Tiene dos dígitos")
        else
            // Condicional para evaluar si el número tiene tres cifras
            if (valor < 1000)
                // Imprime que el número tiene tres dígitos
                println("Tiene tres dígitos")
            else
                // Si el valor no está dentro del rango válido, imprime un mensaje de error
                println("Error en la entrada de datos.")
    }

```



```

Ingrese un valor entero con 1,2 o 3 cifras:34
Tiene dos dígitos

```

**Documentación:** Este programa en Kotlin solicita al usuario ingresar un número entero de hasta tres cifras. A través de una serie de evaluaciones condicionales if-else, determina si el número tiene una, dos o tres cifras, y muestra el resultado en la consola. Si el valor ingresado supera las tres cifras, imprime un mensaje de error indicando que la entrada no es válida. Utiliza la función `readln()` para capturar la entrada del usuario y la convierte a entero con `toInt()`.

### Proyecto 22:

// Definición de la función principal que actúa como punto de entrada al programa

```

fun main(parametro: Array<String>) {

    // Solicita al usuario ingresar la cantidad total de preguntas del examen
    print("Ingrese la cantidad total de preguntas del examen:")

    // Lee la entrada del usuario y la convierte en un entero
    val totalPreguntas = readln().toInt()

```

```

// Solicita al usuario ingresar la cantidad de preguntas contestadas correctamente
print("Ingrese la cantidad total de preguntas contestadas correctamente:")

// Lee la entrada del usuario y la convierte en un entero
val totalCorrectas = readln().toInt()

// Calcula el porcentaje de respuestas correctas
val porcentaje = totalCorrectas * 100 / totalPreguntas

// Clasifica el desempeño según el porcentaje calculado
if (porcentaje >= 90)
    // Si el porcentaje es mayor o igual a 90, imprime "Nivel máximo"
    println("Nivel máximo")
else
    // Si el porcentaje está entre 75 y 89, imprime "Nivel medio"
    if (porcentaje >= 75)
        System.out.print("Nivel medio")
    else
        // Si el porcentaje está entre 50 y 74, imprime "Nivel regular"
        if (porcentaje >= 50)
            println("Nivel regular")
        else
            // Si el porcentaje es menor a 50, imprime "Fuera de nivel"
            println("Fuera de nivel")
}

```

```

Ingrese la cantidad total de preguntas del examen:5
Ingrese la cantidad total de preguntas contestadas correctamente:4
Nivel medio

```

**Documentación:** Este programa en Kotlin solicita al usuario ingresar la cantidad total de preguntas de un examen y el número de preguntas contestadas correctamente. Calcula el porcentaje de

respuestas correctas y, mediante una serie de evaluaciones condicionales if-else, clasifica el desempeño en cuatro niveles: "Nivel máximo" si el porcentaje es mayor o igual a 90, "Nivel medio" si está entre 75 y 89, "Nivel regular" si está entre 50 y 74, y "Fuera de nivel" si es menor al 50. Utiliza la función `readln()` para capturar las entradas y las convierte a enteros con `toInt()`.

### Proyecto 23:

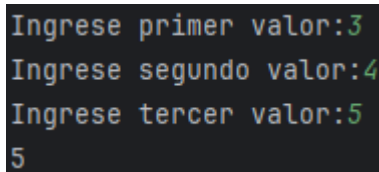
// Definición de la función principal que actúa como punto de entrada al programa

```
fun main(parametro: Array<String>) {  
    // Solicita al usuario ingresar el primer valor  
    print("Ingrese primer valor:")  
    // Lee la entrada del usuario y la convierte en un entero  
    val num1 = readLine()!!.toInt()  
  
    // Solicita al usuario ingresar el segundo valor  
    print("Ingrese segundo valor:")  
    // Lee la entrada del usuario y la convierte en un entero  
    val num2 = readLine()!!.toInt()  
  
    // Solicita al usuario ingresar el tercer valor  
    print("Ingrese tercer valor:")  
    // Lee la entrada del usuario y la convierte en un entero  
    val num3 = readLine()!!.toInt()  
  
    // Condicional para determinar el mayor valor entre los tres números  
    if (num1 > num2 && num1 > num3)  
        // Si num1 es mayor que num2 y num3, imprime num1  
        print(num1)  
    else  
        // Si num1 no es el mayor, verifica si num2 es mayor que num3  
        if (num2 > num3)  
            // Si num2 es mayor que num3, imprime num2
```

```

        print(num2)
    else
        // Si num3 es el mayor, imprime num3
        print(num3)
    }

```



```

Ingrese primer valor:3
Ingrese segundo valor:4
Ingrese tercer valor:5
5

```

**Documentación:** Este programa en Kotlin solicita al usuario ingresar tres valores enteros desde la consola y determina cuál de ellos es el mayor. Utiliza la función `readLine()` para capturar las entradas y las convierte a enteros con `toInt()`. Emplea una estructura condicional `if-else` con operadores lógicos para comparar los valores, asegurándose de identificar y mostrar en la consola el valor más alto entre los tres.

#### Proyecto 24:

// Definición de la función principal que actúa como punto de entrada al programa

```

fun main(parametro: Array<String>) {
    // Solicita al usuario ingresar el día de la fecha
    print("Ingrese día:")

    // Lee la entrada del usuario y la convierte en un entero
    val dia = readLine()!!.toInt()

    // Solicita al usuario ingresar el mes de la fecha
    print("Ingrese mes:")

    // Lee la entrada del usuario y la convierte en un entero
    val mes = readLine()!!.toInt()

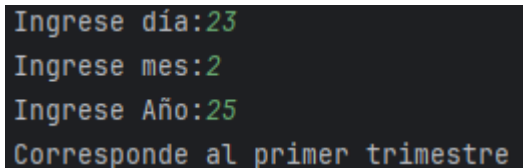
    // Solicita al usuario ingresar el año de la fecha
    print("Ingrese Año:")

    // Lee la entrada del usuario y la convierte en un entero
    val año = readLine()!!.toInt()

```



```
// Condicional para evaluar si el mes ingresado corresponde al primer trimestre
if (mes == 1 || mes == 2 || mes == 3)
    // Si el mes es enero, febrero o marzo, imprime el mensaje correspondiente
    print("Corresponde al primer trimestre")
}
```



```
Ingrese día:23
Ingrese mes:2
Ingrese Año:25
Corresponde al primer trimestre
```

**Documentación:** Este programa en Kotlin permite al usuario ingresar una fecha (día, mes y año) desde la consola y determina si el mes ingresado corresponde al primer trimestre del año. Utiliza la función `readLine()` para capturar los valores y los convierte a enteros con `toInt()`. Mediante una condición `if` con operadores lógicos `||`, evalúa si el mes es enero, febrero o marzo, mostrando un mensaje en la consola si la fecha pertenece al primer trimestre.

#### Proyecto 25:

// Definición de la función principal que actúa como punto de entrada al programa

```
fun main(parametro: Array<String>) {
    // Solicita al usuario ingresar el día
    print("Ingrese día:")

    // Lee la entrada del usuario y la convierte en un entero
    val dia = readln().toInt()

    // Solicita al usuario ingresar el mes
    print("Ingrese mes:")

    // Lee la entrada del usuario y la convierte en un entero
    val mes = readln().toInt()

    // Solicita al usuario ingresar el año
    print("Ingrese Año:")

    // Lee la entrada del usuario y la convierte en un entero
```

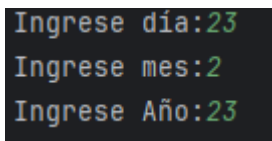
```

val año = readln().toInt()

// Condicional para evaluar si la fecha ingresada corresponde a Navidad
if (mes == 12 && día == 25)

    // Si el mes es 12 y el día es 25, imprime el mensaje correspondiente
    print("La fecha ingresada corresponde a navidad.")
}

```



```

Ingrese día:23
Ingrese mes:2
Ingrese Año:23

```

**Documentación:** Este programa en Kotlin solicita al usuario ingresar una fecha (día, mes y año) y verifica si la fecha ingresada corresponde al 25 de diciembre, que se considera Navidad. Utiliza la función `readln()` para capturar las entradas desde la consola y las convierte a enteros con `toInt()`. Mediante una condición `if`, evalúa si el mes es igual a 12 y el día es igual a 25, mostrando en la consola un mensaje indicando que la fecha corresponde a Navidad.

#### Proyecto 26:

// Definición de la función principal que actúa como punto de entrada al programa

```

fun main(parametro: Array<String>) {

    // Solicita al usuario ingresar el primer valor
    print("Ingrese primer valor:")

    // Lee la entrada del usuario y la convierte en un entero
    val valor1 = readln().toInt()

    // Solicita al usuario ingresar el segundo valor
    print("Ingrese segundo valor:")

    // Lee la entrada del usuario y la convierte en un entero
    val valor2 = readln().toInt()

    // Solicita al usuario ingresar el tercer valor
    print("Ingrese tercer valor:")
}

```

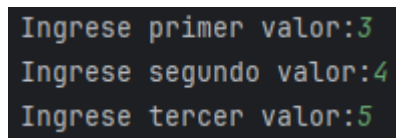
```

// Lee la entrada del usuario y la convierte en un entero
val valor3 = readln().toInt()

// Condicional para verificar si los tres valores son iguales
if (valor1 == valor2 && valor1 == valor3) {
    // Calcula el cubo del valor si los tres valores son iguales
    val cubo = valor1 * valor1 * valor3

    // Imprime el resultado del cálculo con un mensaje
    print("El cubo de $valor1 es $cubo")
}
}

```



```

Ingrese primer valor:3
Ingrese segundo valor:4
Ingrese tercer valor:5

```

**Documentación:** Este programa en Kotlin solicita al usuario ingresar tres valores enteros desde la consola y verifica si todos los valores ingresados son iguales. Si los tres valores son idénticos, calcula el cubo del valor ingresado y lo muestra en la consola junto con un mensaje explicativo. Utiliza la función `readln()` para capturar las entradas del usuario y las convierte a enteros con `toInt()`. Emplea una condición `if` con operadores lógicos para la validación.

#### Proyecto 27:

```

// Definición de la función principal que actúa como punto de entrada al programa
fun main(parametro: Array<String>) {
    // Solicita al usuario ingresar el primer valor
    print("Ingrese primer valor:")

    // Lee la entrada del usuario y la convierte en un entero
    val valor1 = readln().toInt()

    // Solicita al usuario ingresar el segundo valor
    print("Ingrese segundo valor:")

    // Lee la entrada del usuario y la convierte en un entero

```

```

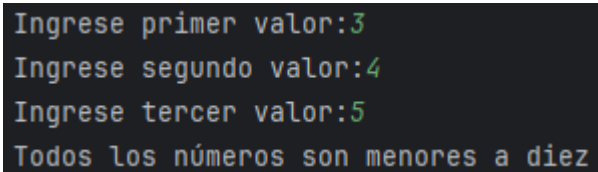
val valor2 = readln().toInt()

// Solicita al usuario ingresar el tercer valor
print("Ingrese tercer valor:")

// Lee la entrada del usuario y la convierte en un entero
val valor3 = readln().toInt()

// Condicional para verificar si los tres valores son menores a diez
if (valor1 < 10 && valor2 < 10 && valor3 < 10)
    // Si todos los valores son menores a diez, imprime el mensaje correspondiente
    print("Todos los números son menores a diez")
}

```



```

Ingrese primer valor:3
Ingrese segundo valor:4
Ingrese tercer valor:5
Todos los números son menores a diez

```

**Documentación:** Este programa en Kotlin solicita al usuario ingresar tres valores enteros desde la consola y verifica si todos los valores son menores a diez. Utiliza la función `readln()` para capturar las entradas y las convierte a enteros con `toInt()`. Mediante una condición `if` con operadores lógicos `&&`, evalúa si los tres valores cumplen la condición y, de ser cierto, muestra un mensaje en la consola indicando que todos los números son menores a diez.

#### Proyecto 28:

```

// Definición de la función principal que actúa como punto de entrada al programa
fun main(parametro: Array<String>) {

    // Solicita al usuario ingresar el primer valor
    print("Ingrese primer valor:")

    // Lee la entrada del usuario y la convierte en un entero
    val valor1 = readln().toInt()

    // Solicita al usuario ingresar el segundo valor
    print("Ingrese segundo valor:")
}

```

```

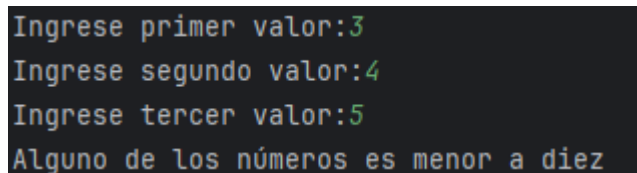
// Lee la entrada del usuario y la convierte en un entero
val valor2 = readln().toInt()

// Solicita al usuario ingresar el tercer valor
print("Ingrese tercer valor:")

// Lee la entrada del usuario y la convierte en un entero
val valor3 = readln().toInt()

// Condicional para verificar si alguno de los valores es menor a diez
if (valor1 < 10 || valor2 < 10 || valor3 < 10)
    // Si al menos un valor es menor a diez, imprime el mensaje correspondiente
    print("Alguno de los números es menor a diez")
}

```



```

Ingrese primer valor:3
Ingrese segundo valor:4
Ingrese tercer valor:5
Alguno de los números es menor a diez

```

**Documentación:** Este programa en Kotlin solicita al usuario ingresar tres valores enteros desde la consola y verifica si al menos uno de ellos es menor a diez. Utiliza la función `readln()` para capturar las entradas del usuario y las convierte a enteros con `toInt()`. Mediante una condición `if` con el operador lógico `||`, evalúa si cualquiera de los tres valores cumple la condición, mostrando un mensaje en la consola si esto se cumple.

#### Proyecto 29:

```

// Definición de la función principal que actúa como punto de entrada al programa
fun main(parametro: Array<String>) {

    // Solicita al usuario ingresar la coordenada x
    print("Ingrese coordenada x:")

    // Lee la entrada del usuario y la convierte en un entero
    val x = readln().toInt()

    // Solicita al usuario ingresar la coordenada y

```

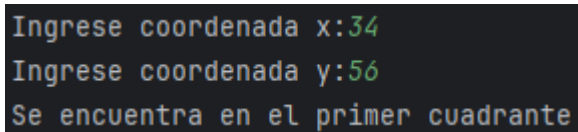
```

print("Ingrese coordenada y:")

// Lee la entrada del usuario y la convierte en un entero
val y = readln().toInt()

// Condicional para determinar en qué cuadrante se encuentra el punto
if (x > 0 && y > 0)
    // Si x es positivo y y es positivo, el punto está en el primer cuadrante
    print("Se encuentra en el primer cuadrante")
else
    // Si x es negativo y y es positivo, el punto está en el segundo cuadrante
    if (x < 0 && y > 0)
        print("Se encuentra en el segundo cuadrante")
    else
        // Si x es negativo y y es negativo, el punto está en el tercer cuadrante
        if (x < 0 && y < 0)
            print("Se encuentra en el tercer cuadrante")
        else
            // Si x es positivo y y es negativo, el punto está en el cuarto cuadrante
            if (x > 0 && y < 0)
                print("Se encuentra en el cuarto cuadrante")
            else
                // Si ninguna condición anterior se cumple, el punto está en un eje
                print("Se encuentra en un eje")
}

```



```

Ingrese coordenada x:34
Ingrese coordenada y:56
Se encuentra en el primer cuadrante

```

**Documentación:** Este programa en Kotlin solicita al usuario ingresar las coordenadas x e y de un punto en el plano cartesiano y determina en qué cuadrante se encuentra el punto. Emplea una serie de condiciones if-else con operadores lógicos para verificar si las coordenadas corresponden al primer,

segundo, tercer o cuarto cuadrante, y muestra el resultado en la consola. Si el punto se encuentra en alguno de los ejes, imprime un mensaje indicando esta situación.

### **Proyecto 30:**

// Definición de la función principal que actúa como punto de entrada al programa

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

```
    // Solicita al usuario ingresar el primer valor
```

```
    print("Ingrese primer valor:")
```

```
    // Lee la entrada del usuario y la convierte en un entero
```

```
    val valor1 = readln().toInt()
```

```
    // Solicita al usuario ingresar el segundo valor
```

```
    print("Ingrese segundo valor:")
```

```
    // Lee la entrada del usuario y la convierte en un entero
```

```
    val valor2 = readln().toInt()
```

```
    // Solicita al usuario ingresar el tercer valor
```

```
    print("Ingrese tercer valor:")
```

```
    // Lee la entrada del usuario y la convierte en un entero
```

```
    val valor3 = readln().toInt()
```

```
    // Usa una expresión condicional para determinar el menor valor
```

```
    val menor = if (valor1 < valor2 && valor1 < valor3) valor1 else if (valor2 < valor3) valor2 else
    valor3
```

```
    // Usa una expresión condicional para determinar el mayor valor
```

```
    val mayor = if (valor1 > valor2 && valor1 > valor3) valor1 else if (valor2 > valor3) valor2 else
    valor3
```

```
    // Imprime el mayor y el menor valor en la consola
```

```
    print("El mayor de la lista es $mayor y el menor $menor")
```

```
}
```

```
Ingrese primer valor:89
Ingrese segundo valor:6
Ingrese tercer valor:5
El mayor de la lista es 89 y el menor 5
```

**Documentación:** Este programa en Kotlin solicita al usuario ingresar tres valores enteros y determina cuál es el mayor y cuál es el menor entre ellos. Utiliza la función `readln()` para capturar las entradas desde la consola y las convierte a enteros con `toInt()`. Mediante expresiones condicionales `if-else`, evalúa las comparaciones necesarias para asignar los valores correspondientes a las variables `mayor` y `menor`. Finalmente, muestra los resultados en la consola indicando cuál es el mayor y cuál es el menor.

### Proyecto 31:

// Definición de la función principal que actúa como punto de entrada al programa

```
fun main(parametro: Array<String>) {
    // Inicializa la variable x con un valor de 1
    var x = 1

    // Bucle while que se ejecuta mientras x sea menor o igual a 100
    while (x <= 100) {
        // Imprime el valor actual de x en la consola
        println(x)
        // Incrementa el valor de x en 1
        x = x + 1
    }
}
```

```
94
95
96
97
98
99
100
```



**Documentación:** Este programa en Kotlin imprime todos los números enteros del 1 al 100, inclusive. Utiliza una estructura de control while para iterar mientras la variable x cumpla la condición  $x \leq 100$ . Dentro del bucle, imprime el valor actual de x en la consola y luego lo incrementa en 1 en cada iteración hasta que la condición deje de cumplirse.

### Proyecto 32:

// Definición de la función principal que actúa como punto de entrada al programa

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

```
    // Solicita al usuario ingresar un valor entero
```

```
    print("Ingrese un valor:")
```

```
    // Lee la entrada del usuario y la convierte en un entero
```

```
    val valor = readLine()!!.toInt()
```

```
    // Inicializa la variable x con el valor de 1
```

```
    var x = 1
```

```
    // Bucle while que se ejecuta mientras x sea menor o igual al valor ingresado
```

```
    while (x <= valor) {
```

```
        // Imprime el valor actual de x en la consola
```

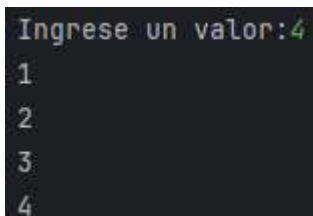
```
        println(x)
```

```
        // Incrementa el valor de x en 1
```

```
        x = x + 1
```

```
    }
```

```
}
```



```
Ingrese un valor:4
1
2
3
4
```

**Documentación:** Este programa en Kotlin solicita al usuario ingresar un número entero y utiliza un bucle while para imprimir todos los números enteros desde 1 hasta el valor ingresado, inclusive. La función `readLine()` captura la entrada del usuario y la convierte a entero con `toInt()`. Dentro del bucle,

se imprime el valor actual de la variable x en cada iteración, incrementándola en 1 hasta que cumple con la condición de salida  $x \leq \text{valor}$ .

### Proyecto 33:

// Definición de la función principal que actúa como punto de entrada al programa

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

```
    // Inicializa la variable x para controlar el bucle
```

```
    var x = 1
```

```
    // Inicializa la variable suma para acumular los valores ingresados
```

```
    var suma = 0
```

```
    // Bucle while que se ejecuta mientras x sea menor o igual a 10
```

```
    while (x <= 10) {
```

```
        // Solicita al usuario ingresar un valor entero
```

```
        print("Ingrese un valor:")
```

```
        // Lee la entrada del usuario y la convierte en un entero
```

```
        val valor = readLine()!!.toInt()
```

```
        // Acumula el valor ingresado en la variable suma
```

```
        suma = suma + valor
```

```
        // Incrementa la variable x en 1 para controlar la cantidad de iteraciones
```

```
        x = x + 1
```

```
    }
```

```
    // Imprime la suma total de los 10 valores ingresados
```

```
    println("La suma de los 10 valores ingresados es $suma")
```

```
    // Calcula el promedio dividiendo la suma entre 10
```

```
    val promedio = suma / 10
```

```
    // Imprime el promedio calculado
```

```
    println("El promedio es $promedio")
```

```
}
```

```
Ingrese un valor:8
Ingrese un valor:9
Ingrese un valor:7
Ingrese un valor:6
Ingrese un valor:5
La suma de los 10 valores ingresados es 60
El promedio es 6
```

**Documentación:** Este programa en Kotlin solicita al usuario ingresar diez valores enteros mediante un bucle while. Suma todos los valores ingresados, calcula el promedio de los mismos y presenta ambos resultados en la consola. Utiliza la función `readLine()` para capturar las entradas del usuario, convierte cada valor a entero con `toInt()`, y realiza las operaciones matemáticas correspondientes dentro del bucle.

#### Proyecto 34:

// Definición de la función principal que actúa como punto de entrada al programa

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

```
    // Solicita al usuario ingresar la cantidad total de piezas a procesar
```

```
    print("Cuantas piezas procesará:")
```

```
    // Lee la entrada del usuario y la convierte en un entero
```

```
    val n = readLine()!!.toInt()
```

```
    // Inicializa la variable x para controlar el bucle
```

```
    var x = 1
```

```
    // Inicializa la variable cantidad para contar las piezas aptas
```

```
    var cantidad = 0
```

```
    // Bucle while que se ejecuta para procesar todas las piezas
```

```
    while (x <= n) {
```

```
        // Solicita al usuario ingresar la medida de la pieza
```

```
        print("Ingrese la medida de la pieza:")
```

```
        // Lee la medida de la pieza y la convierte en un número decimal
```

```
        val largo = readLine()!!.toDouble()
```

```
        // Condicional para verificar si la medida está en el rango de aptitud
```

```

        if (largo >= 1.20 && largo <= 1.30)

            // Si la medida es apta, incrementa el contador cantidad

            cantidad = cantidad + 1

            // Incrementa la variable x en 1 para controlar la cantidad de iteraciones

            x = x + 1

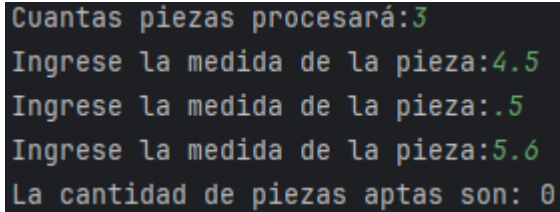
        }

// Imprime la cantidad total de piezas aptas

print("La cantidad de piezas aptas son: $cantidad")

}

```



```

Cuantas piezas procesará:3
Ingrese la medida de la pieza:4.5
Ingrese la medida de la pieza:.5
Ingrese la medida de la pieza:5.6
La cantidad de piezas aptas son: 0

```

**Documentación:** Este programa en Kotlin solicita al usuario ingresar un número de piezas a procesar y la medida de cada una. Evalúa si la longitud de la pieza está dentro del rango de aptitud definido (entre 1.20 y 1.30). Utiliza un bucle while para procesar todas las piezas y, mediante una condición if, incrementa un contador para aquellas que cumplen con los criterios. Finalmente, muestra la cantidad de piezas aptas en la consola.

### Proyecto 35:

```

fun main() {

    // Lee tres valores ingresados por el usuario

    val valor1 = readln().toInt()

    val valor2 = readln().toInt()

    val valor3 = readln().toInt()

    // Determina el menor y mayor valor

    val menor = if (valor1 < valor2 && valor1 < valor3) valor1 else if (valor2 < valor3) valor2 else
valor3

    val mayor = if (valor1 > valor2 && valor1 > valor3) valor1 else if (valor2 > valor3) valor2 else
valor3
}

```

```

// Muestra los resultados

println("Mayor: $mayor, Menor: $menor")

}

```

**Documentación:** Este programa solicita al usuario ingresar tres valores enteros y determina el mayor y menor de ellos. Utiliza expresiones condicionales para realizar las comparaciones y presenta los resultados en la consola.

#### Proyecto 36:

```

fun main() {

    var x = 1 // Inicializa la variable de control

    while (x <= 100) { // Itera mientras x sea menor o igual a 100

        println(x) // Imprime el valor actual de x

        x++ // Incrementa la variable de control

    }

}

```

**Documentación:** Este programa imprime todos los números enteros desde 1 hasta 100 utilizando un bucle while. La variable de control se incrementa en cada iteración y los valores se muestran en la consola.

#### Proyecto 37:

```

fun main() {

    print("Ingresa un valor: ") // Solicita al usuario un número entero

    val valor = readln().toInt() // Convierte la entrada a entero

    var x = 1 // Inicializa la variable de control

    while (x <= valor) { // Itera mientras x sea menor o igual al valor

        println(x) // Imprime el valor actual de x

        x++ // Incrementa la variable de control

    }

}

```

**Documentación:** Este programa solicita al usuario un número entero y utiliza un bucle while para imprimir desde 1 hasta el número ingresado, inclusive.

### Proyecto 38:

```
fun main() {  
    var x = 1 // Inicializa el contador  
    var suma = 0 // Acumulador para la suma  
    while (x <= 10) { // Bucle que se ejecuta 10 veces  
        print("Ingresa un valor: ") // Solicita al usuario un valor  
        val valor = readln().toInt() // Convierte la entrada a entero  
        suma += valor // Suma el valor ingresado al acumulador  
        x++ // Incrementa el contador  
    }  
    println("La suma de los valores es $suma") // Muestra la suma total  
    val promedio = suma / 10 // Calcula el promedio  
    println("El promedio es $promedio") // Muestra el promedio  
}
```

**Documentación:** Este programa solicita al usuario ingresar diez valores enteros, calcula la suma de estos valores y también el promedio, mostrando ambos resultados al finalizar.

Código comentado

```
fun main() { var x = 1 // Inicializa el contador var suma = 0 // Acumulador para la suma while (x <= 10) { // Bucle que se ejecuta 10 veces print("Ingresa un valor: ") // Solicita al usuario un valor val valor = readln().toInt() // Convierte la entrada a entero suma += valor // Suma el valor ingresado al acumulador x++ // Incrementa el contador } println("La suma de los valores es $suma") // Muestra la suma total val promedio = suma / 10 // Calcula el promedio println("El promedio es $promedio") // Muestra el promedio }
```

### Documentación

Este programa evalúa si un número ingresado por el usuario es positivo, negativo o cero, utilizando una estructura condicional simple.

### Proyecto 39:

```
fun main(parametro: Array<String>) {  
    // Inicializa la variable con el primer múltiplo de 8  
    var mult8 = 8
```

```
// Bucle que se ejecuta mientras mult8 sea menor o igual a 500
while (mult8 <= 500) {
    // Imprime el valor actual de mult8 seguido de un guion
    print("$mult8 -")

    // Incrementa mult8 en 8 para pasar al siguiente múltiplo
    mult8 = mult8 + 8
}
}
```

**Documentación:** Este programa en Kotlin genera y muestra todos los múltiplos de 8 desde 8 hasta 500 inclusive. Utiliza un bucle while para iterar, comenzando con el valor inicial de 8 y aumentando en pasos de 8 en cada ciclo. Los múltiplos se imprimen en la consola separados por un guion.

#### Proyecto 40:

```
fun main(parametro: Array<String>) {
    var x = 1 // Inicializa el contador para la primera lista
    var suma1 = 0 // Variable para acumular la suma de la primera lista

    // Solicita los valores de la primera lista
    println("Ingreso de la primer lista de valores")
    while (x <= 5) { // Itera para ingresar 5 valores
        print("Ingresa valor:") // Solicita un valor al usuario
        val valor = readln().toInt() // Convierte la entrada a entero
        suma1 += valor // Acumula el valor en suma1
        x++ // Incrementa el contador
    }

    // Reinicia el contador para la segunda lista
    println("Ingreso de la segunda lista de valores")
    x = 1

    var suma2 = 0 // Variable para acumular la suma de la segunda lista
```

```

// Solicita los valores de la segunda lista
while (x <= 5) { // Itera para ingresar otros 5 valores
    print("Ingresa valor:") // Solicita un valor al usuario
    val valor = readln().toInt() // Convierte la entrada a entero
    suma2 += valor // Acumula el valor en suma2
    x++ // Incrementa el contador
}

// Compara las sumas de ambas listas
if (suma1 > suma2)
    print("Lista 1 mayor.") // Primera lista tiene mayor suma
else if (suma2 > suma1)
    print("Lista2 mayor.") // Segunda lista tiene mayor suma
else
    print("Listas iguales.") // Ambas listas tienen igual suma
}

```

**Documentación:** Este programa en Kotlin permite al usuario ingresar dos listas de valores, cada una con cinco elementos, y calcula la suma de los valores de cada lista. Compara las sumas y determina si la primera lista, la segunda lista o ambas tienen igual valor total, mostrando el resultado correspondiente en la consola.

#### **Proyecto 41:**

```

fun main(parametro: Array<String>) {
    var pares = 0 // Inicializa el contador de números pares
    var impares = 0 // Inicializa el contador de números impares

    // Solicita al usuario la cantidad de números que desea ingresar
    print("Cuantos números ingresará:")
    val n = readln().toInt() // Convierte la entrada a entero

```



```

var x = 1 // Inicializa el contador del bucle

// Bucle que se ejecuta n veces
while (x <= n) {
    print("Ingrese el valor:") // Solicita un número al usuario
    val valor = readln().toInt() // Convierte la entrada a entero

    // Condicional para verificar si el número es par o impar
    if (valor % 2 == 0)
        pares = pares + 1 // Incrementa el contador de pares si el número es par
    else
        impares = impares + 1 // Incrementa el contador de impares si el número es impar

    x = x + 1 // Incrementa el contador del bucle
}

// Imprime la cantidad total de números pares
println("Cantidad de pares: $pares")

// Imprime la cantidad total de números impares
println("Cantidad de impares: $impares")
}

```

**Documentación:** Este programa en Kotlin solicita al usuario ingresar una cantidad n de números enteros y clasifica los valores ingresados en números pares o impares. Utiliza un bucle while para iterar n veces y una estructura condicional if-else para contar cuántos son pares y cuántos son impares. Al finalizar, muestra las cantidades respectivas en la consola.

#### Proyecto 42:

```

fun main(parametro: Array<String>) {
    do {
        // Solicita al usuario ingresar un valor entre 0 y 999
        print("Ingrese un valor comprendido entre 0 y 999:")
        val valor = readLine()!!.toInt() // Convierte la entrada a entero
    }
}

```

```

// Condicional para evaluar la cantidad de dígitos del valor ingresado
if (valor < 10)
    println("El valor ingresado tiene un dígito") // Para valores menores a 10
else if (valor < 100)
    println("El valor ingresado tiene dos dígitos") // Para valores entre 10 y 99
else
    println("El valor ingresado tiene tres dígitos") // Para valores entre 100 y 999

} while (valor != 0) // Repite el bucle mientras el valor ingresado sea diferente de 0
}

```

**Documentación:** Este programa en Kotlin permite al usuario ingresar repetidamente valores comprendidos entre 0 y 999. Para cada valor, evalúa cuántos dígitos tiene (uno, dos o tres) y lo informa en la consola. El programa utiliza un bucle do-while que se ejecuta hasta que el usuario ingresa el valor 0.

#### **Proyecto 43:**

```

fun main(parametro: Array<String>) {
    var cant = 0 // Inicializa el contador de cantidad de valores
    var suma = 0 // Inicializa la suma acumulada de los valores

    // Bucle do-while para solicitar valores hasta que el usuario ingrese 0
    do {
        print("Ingrese un valor (0 para finalizar):") // Solicita un valor al usuario
        val valor = readLine()!!.toInt() // Convierte la entrada a entero
        if (valor != 0) { // Verifica si el valor ingresado es distinto de cero
            suma += valor // Suma el valor a la acumulada
            cant++ // Incrementa el contador de valores ingresados
        }
    } while (valor != 0) // Repite el bucle mientras el valor sea distinto de cero
}

```

```
// Verifica si se ingresaron valores para calcular el promedio

if (cant != 0) {
    val promedio = suma / cant // Calcula el promedio
    print("El promedio de los valores ingresados es: $promedio") // Muestra el promedio
} else
    print("No se ingresaron valores.") // Mensaje si no se ingresaron datos válidos
}
```

**Documentación:** Este programa en Kotlin permite al usuario ingresar varios valores enteros y calcula el promedio de los valores ingresados. El proceso termina cuando el usuario ingresa el valor 0. Utiliza un bucle do-while para recopilar los valores y realizar el cálculo de manera interactiva. Si no se ingresan valores distintos de cero, se muestra un mensaje indicando que no se ingresaron datos.

#### Proyecto 44:

```
fun main(parametro: Array<String>) {
    var cant1 = 0 // Contador para piezas con peso superior a 10.2
    var cant2 = 0 // Contador para piezas aptas (peso entre 9.8 y 10.2)
    var cant3 = 0 // Contador para piezas con peso inferior a 9.8

    // Bucle que solicita pesos hasta que se ingrese 0
    do {
        print("Ingrese el peso de la pieza (0 para finalizar):") // Solicita peso
        val peso = readLine()!!.toDouble() // Convierte la entrada a número decimal

        // Clasifica las piezas según el peso
        if (peso > 10.2)
            cant1++ // Incrementa el contador de piezas con peso superior a 10.2
        else if (peso >= 9.8)
            cant2++ // Incrementa el contador de piezas aptas
        else if (peso > 0)
            cant3++ // Incrementa el contador de piezas con peso inferior a 9.8
    } while (peso != 0)
}
```

```

    } while (peso != 0.0) // Repite el bucle hasta que se ingrese 0

    // Imprime los resultados
    println("Piezas aptas: $cant2")
    println("Piezas con un peso superior a 10.2: $cant1")
    println("Piezas con un peso inferior a 9.8: $cant3")

    // Calcula y muestra la cantidad total de piezas procesadas
    val suma = cant1 + cant2 + cant3

    println("Cantidad total de piezas procesadas: $suma")
}

```

**Documentación:** Este programa en Kotlin permite registrar el peso de varias piezas y clasifica cada una en tres categorías: piezas con peso superior a 10.2, piezas aptas (peso entre 9.8 y 10.2 inclusive), y piezas con peso inferior a 9.8. Utiliza un bucle do-while que finaliza al ingresar el valor 0. Finalmente, imprime la cantidad de piezas en cada categoría y el total de piezas procesadas.

#### Proyecto 45:

```

fun main(parametro: Array<String>) {
    var suma = 0 // Inicializa la suma acumulada en cero

    // Bucle do-while que solicita valores hasta que se ingrese 9999
    do {
        print("Ingrese un valor (finalizar con 9999):") // Solicita un valor al usuario
        val valor = readln().toInt() // Convierte la entrada a entero
        if (valor != 9999) // Verifica si el valor es distinto de 9999
            suma += valor // Acumula el valor en la suma
    } while (valor != 9999) // Repite el bucle mientras el valor sea distinto de 9999

    // Imprime la suma acumulada
    println("El valor acumulado es $suma")
}

```

```

// Evalúa si la suma acumulada es cero, positiva o negativa

if (suma == 0)

    println("El valor acumulado es cero.") // Mensaje si la suma es cero

else if (suma > 0)

    println("El valor acumulado es positivo.") // Mensaje si la suma es positiva

else

    println("El valor acumulado es negativo") // Mensaje si la suma es negativa

}

```

**Documentación:** Este programa en Kotlin permite al usuario ingresar valores enteros y acumula su suma. El proceso se detiene cuando el usuario ingresa el valor 9999, el cual no se acumula. Después de finalizar, el programa evalúa si la suma acumulada es cero, positiva o negativa y muestra el resultado correspondiente en la consola.

#### **Proyecto 46:**

```

fun main(parametro: Array<String>) {

    var suma = 0.0 // Inicializa el acumulador para los saldos acreedores

    // Bucle que solicita datos mientras el número de cuenta sea positivo o cero
    do {

        print("Ingrese número de cuenta:") // Solicita el número de cuenta

        val cuenta = readln().toInt() // Convierte la entrada a entero

        if (cuenta >= 0) { // Verifica si la cuenta es válida (número positivo o cero)

            print("Ingrese saldo:") // Solicita el saldo de la cuenta

            val saldo = readln().toDouble() // Convierte el saldo a un número decimal

            // Determina el tipo de saldo y realiza acciones correspondientes

            if (saldo > 0) {

                println("Saldo Acreedor.") // Indica que el saldo es acreedor

                suma += saldo // Acumula el saldo acreedor

            } else if (saldo < 0)

```

```

        println("Saldo Deudor.") // Indica que el saldo es deudor
    else
        println("Saldo Nulo.") // Indica que el saldo es nulo
    }
} while (cuenta >= 0) // Repite mientras el número de cuenta sea válido

// Muestra la suma total de los saldos acreedores
println("Total de saldos Acreedores: $suma")
}

```

**Documentación:** Este programa en Kotlin permite al usuario registrar cuentas y sus respectivos saldos. Mientras el número de cuenta sea mayor o igual a 0, solicita el saldo asociado e informa si es acreedor (positivo), deudor (negativo) o nulo (igual a 0). Además, acumula los saldos acreedores y muestra el total al finalizar. El proceso termina cuando se ingresa un número de cuenta negativo.

#### Proyecto 47:

```

fun main(parametro: Array<String>) {
    // Bucle for que recorre los números del 1 al 100
    for (i in 1..100)
        println(i) // Imprime el valor actual de i en cada iteración
}

```

**Documentación:** Este programa en Kotlin utiliza un bucle for para imprimir los números enteros del 1 al 100, inclusive. La estructura del bucle utiliza un rango definido (1..100) para iterar automáticamente sobre cada número dentro de ese rango y mostrarlo en la consola.

#### Proyecto 48:

```

fun main(parametro: Array<String>) {
    var suma = 0 // Inicializa la variable suma en 0 para acumular los valores

    // Bucle for que recorre 10 iteraciones
    for (i in 1..10) {
        print("Ingrese un valor:") // Solicita al usuario un valor
        val valor = readLine()!!.toInt() // Convierte la entrada a entero
    }
}

```

```

        suma += valor // Suma el valor ingresado a la variable suma
    }

    // Imprime la suma total de los valores ingresados
    println("La suma de los valores ingresados es $suma")

    // Calcula el promedio dividiendo la suma entre 10
    val promedio = suma / 10

    // Imprime el promedio de los valores ingresados
    println("Su promedio es $promedio")
}

```

**Documentación:** Este programa en Kotlin solicita al usuario ingresar diez valores enteros y calcula la suma total y el promedio de dichos valores. Utiliza un bucle for que itera 10 veces, acumulando cada valor ingresado por el usuario. Al finalizar, muestra en la consola los resultados correspondientes.

#### **Proyecto 49:**

```

fun main(parametro: Array<String>) {

    var aprobados = 0 // Contador para alumnos aprobados
    var reprobados = 0 // Contador para alumnos reprobados

    // Bucle for para ingresar las notas de 10 estudiantes
    for (i in 1..10) {

        print("Ingrese nota:") // Solicita la nota del estudiante
        val nota = readLine()!!.toInt() // Convierte la entrada a entero

        // Condicional para clasificar la nota como aprobada o reprobada
        if (nota >= 7)

            aprobados++ // Incrementa el contador de aprobados si la nota es mayor o igual a 7
        else

            reprobados++ // Incrementa el contador de reprobados si la nota es menor a 7
    }
}

```

```

    }

    // Imprime la cantidad de alumnos aprobados y reprobados
    println("Cantidad de alumnos con notas mayores o iguales a 7: $aprobados")
    println("Cantidad de alumnos con notas menores a 7: $reprobados")
}

```

**Documentación:** Este programa en Kotlin solicita al usuario ingresar las notas de 10 estudiantes y clasifica dichas notas en dos categorías: aprobados (notas mayores o iguales a 7) y reprobados (notas menores a 7). Utiliza un bucle for para iterar 10 veces y una estructura condicional if-else para incrementar los contadores respectivos. Al finalizar, muestra la cantidad de aprobados y reprobados en la consola.

#### Proyecto 50:

```

fun main(parametro: Array<String>) {
    var mult3 = 0 // Contador para múltiplos de 3
    var mult5 = 0 // Contador para múltiplos de 5
    var mult9 = 0 // Contador para múltiplos de 8

    // Bucle for que recorre los números del 1 al 10,000
    for (i in 1..10000) {
        if (i % 3 == 0) // Verifica si el número es divisible por 3
            mult3++ // Incrementa el contador de múltiplos de 3
        if (i % 5 == 0) // Verifica si el número es divisible por 5
            mult5++ // Incrementa el contador de múltiplos de 5
        if (i % 8 == 0) // Verifica si el número es divisible por 8
            mult9++ // Incrementa el contador de múltiplos de 8
    }

    // Imprime los resultados
    println("Cantidad de múltiplos de 3: $mult3")
    println("Cantidad de múltiplos de 5: $mult5")
}

```



```
println("Cantidad de múltiplos de 9: $mult9")
}
```

**Documentación:** Este programa en Kotlin cuenta la cantidad de números entre 1 y 10,000 que son múltiplos de 3, 5 y 8 respectivamente. Utiliza un bucle for para iterar por cada número en el rango, comprobando con operadores de módulo (%) si el número es divisible por 3, 5 o 8. Finalmente, imprime la cantidad de múltiplos encontrados para cada caso.

### Proyecto 51:

```
fun main(parametros: Array<String>) {
    var cant = 0 // Inicializa el contador para los números pares

    // Solicita al usuario la cantidad de valores a ingresar
    print("Cuántos valores ingresará para analizar:")
    val cantidad = readLine()!!.toInt() // Convierte la entrada a entero

    // Bucle for que se ejecuta la cantidad de veces especificada por el usuario
    for (i in 1..cantidad) {
        print("Ingrese valor:") // Solicita el ingreso de un valor
        val valor = readLine()!!.toInt() // Convierte la entrada a entero

        // Verifica si el valor es par (divisible entre 2)
        if (valor % 2 == 0)
            cant++ // Incrementa el contador si el valor es par
    }

    // Imprime la cantidad total de números pares encontrados
    println("Cantidad de pares: $cant")
}
```

**Documentación:** Este programa en Kotlin solicita al usuario indicar cuántos valores desea analizar. Luego, el programa evalúa si cada valor ingresado es par (divisible entre 2) utilizando un bucle for y una condición if. Al finalizar, muestra la cantidad total de números pares encontrados.

### Proyecto 52:

```
fun main(parametro: Array<String>) {  
    var cantidad = 0 // Contador para triángulos con superficie superior a 12  
  
    // Solicita al usuario la cantidad de triángulos a procesar  
    print("Cuántos triángulos procesará:")  
    val n = readln().toInt() // Convierte la entrada a entero  
  
    // Bucle for que itera para procesar cada triángulo  
    for (i in 1..n) {  
        print("Ingrese el valor de la base:") // Solicita la base del triángulo  
        val base = readln().toInt() // Convierte la entrada a entero  
  
        print("Ingrese el valor de la altura:") // Solicita la altura del triángulo  
        val altura = readln().toInt() // Convierte la entrada a entero  
  
        // Calcula la superficie del triángulo  
        val superficie = base * altura / 2  
        println("La superficie es de $superficie") // Muestra la superficie calculada  
  
        // Incrementa el contador si la superficie es mayor a 12  
        if (superficie > 12)  
            cantidad++  
    }  
  
    // Imprime la cantidad de triángulos con superficie superior a 12  
    print("La cantidad de triángulos con superficie superior a 12 son: $cantidad")  
}
```

**Documentación:** Este programa en Kotlin solicita al usuario la cantidad de triángulos a procesar, luego solicita las dimensiones (base y altura) de cada triángulo. Calcula la superficie de cada uno mediante la fórmula  $(base * altura) / 2$  y cuenta cuántos tienen una superficie mayor a 12. Al finalizar, imprime las superficies individuales y la cantidad de triángulos que superan el umbral.

#### Proyecto 53:

```
fun main(parametro: Array<String>) {  
    var suma = 0 // Inicializa la variable suma en 0  
  
    // Bucle for para solicitar 10 valores  
    for (i in 1..10) {  
        print("Ingresa un valor entero:") // Solicita un valor entero al usuario  
        val valor = readln().toInt() // Convierte la entrada a entero  
  
        // Verifica si el índice de iteración es mayor a 5  
        if (i > 5)  
            suma += valor // Suma los valores ingresados a partir de la sexta iteración  
    }  
  
    // Imprime la suma de los últimos 5 valores  
    print("La suma de los últimos 5 valores es: $suma")  
}
```

**Documentación:** Este programa en Kotlin solicita al usuario ingresar diez valores enteros y calcula la suma de los últimos cinco valores ingresados. Utiliza un bucle for para recorrer las 10 iteraciones y una condición if para identificar los valores ingresados a partir de la sexta iteración. Finalmente, imprime la suma de esos valores en la consola.

#### Proyecto 54:

```
fun main(parametro: Array<String>) {  
    // Bucle for que genera múltiplos de 5 desde 5 hasta 50  
    for (tabla5 in 5..50 step 5)  
        println(tabla5) // Imprime el valor actual del múltiplo de 5  
}
```

**Documentación:** Este programa en Kotlin utiliza un bucle for para imprimir la tabla del 5 desde 5 hasta 50, inclusive. El rango definido en el bucle (5..50) utiliza el parámetro step para incrementar de cinco en cinco, lo que permite generar los múltiplos de forma eficiente y clara.

**Proyecto 55:**

```
fun main(argumento: Array<String>) {  
    // Solicita al usuario un valor entre 1 y 10  
  
    print("Ingrese un valor entre 1 y 10:")  
  
    val valor = readln().toInt() // Convierte la entrada a entero  
  
  
    // Bucle for que genera múltiplos del valor ingresado  
    for (i in valor..valor * 12 step valor)  
        println(i) // Imprime el múltiplo actual en cada iteración  
}
```

**Documentación:** Este programa en Kotlin solicita al usuario ingresar un valor entre 1 y 10, y luego genera una serie de múltiplos del valor ingresado. Utiliza un bucle for con un rango desde el valor hasta el producto del valor por 12 (valor\*12), incrementando por el propio valor (step valor). Los números generados se imprimen en la consola.

**Proyecto 56:**

```
fun main(argumento: Array<String>) {  
    var cant1 = 0 // Contador para triángulos equiláteros  
    var cant2 = 0 // Contador para triángulos isósceles  
    var cant3 = 0 // Contador para triángulos escalenos  
  
  
    // Solicita al usuario la cantidad de triángulos a procesar  
    print("Ingrese la cantidad de triángulos:")  
  
    val n = readln().toInt() // Convierte la entrada a entero  
  
  
    // Bucle for para procesar cada triángulo  
    for (i in 1..n) {  
        print("Ingrese lado 1:") // Solicita la medida del primer lado
```

```

val lado1 = readln().toInt() // Convierte la entrada a entero

println("Ingrese lado 2:") // Solicita la medida del segundo lado
val lado2 = readln().toInt() // Convierte la entrada a entero

println("Ingrese lado 3:") // Solicita la medida del tercer lado
val lado3 = readln().toInt() // Convierte la entrada a entero

// Clasifica el triángulo según las medidas de sus lados
if (lado1 == lado2 && lado1 == lado3) {
    println("Es un triángulo equilátero.") // Triángulo con todos los lados iguales
    cant1++ // Incrementa el contador de triángulos equiláteros
} else if (lado1 == lado2 || lado1 == lado3 || lado2 == lado3) {
    println("Es un triángulo isósceles.") // Triángulo con dos lados iguales
    cant2++ // Incrementa el contador de triángulos isósceles
} else {
    println("Es un triángulo escaleno.") // Triángulo con todos los lados diferentes
    cant3++ // Incrementa el contador de triángulos escalenos
}
}

// Imprime la cantidad de triángulos clasificados en cada categoría
println("Cantidad de triángulos equiláteros: $cant1")
println("Cantidad de triángulos isósceles: $cant2")
println("Cantidad de triángulos escalenos: $cant3")
}

```

**Documentación:** Este programa en Kotlin clasifica un conjunto de triángulos en tres tipos: equiláteros (todos los lados iguales), isósceles (dos lados iguales) y escalenos (todos los lados diferentes). Solicita al usuario la cantidad de triángulos a procesar y las medidas de sus lados,

contando cuántos triángulos pertenecen a cada categoría. Finalmente, imprime la cantidad de triángulos por tipo.

**Proyecto 57:**

```
fun main(parametro: Array<String>) {  
    var cant1 = 0 // Contador para puntos en el primer cuadrante  
    var cant2 = 0 // Contador para puntos en el segundo cuadrante  
    var cant3 = 0 // Contador para puntos en el tercer cuadrante  
    var cant4 = 0 // Contador para puntos en el cuarto cuadrante  
  
    // Solicita al usuario la cantidad de puntos a procesar  
    print("Cantidad de puntos a ingresar:")  
    val cantidad = readln().toInt() // Convierte la entrada a entero  
  
    // Bucle for para procesar la cantidad de puntos especificados  
    for (i in 1..cantidad) {  
        print("Ingrese coordenada x:") // Solicita la coordenada x del punto  
        val x = readln().toInt() // Convierte x a entero  
  
        print("Ingrese coordenada y:") // Solicita la coordenada y del punto  
        val y = readln().toInt() // Convierte y a entero  
  
        // Clasifica el punto según el cuadrante al que pertenece  
        if (x > 0 && y > 0) // Primer cuadrante  
            cant1++  
        else if (x < 0 && y > 0) // Segundo cuadrante  
            cant2++  
        else if (x < 0 && y < 0) // Tercer cuadrante  
            cant3++  
        else if (x > 0 && y < 0) // Cuarto cuadrante
```

```

        cant4++
    }

    // Imprime la cantidad de puntos en cada cuadrante
    println("Cantidad de puntos en el primer cuadrante: $cant1")
    println("Cantidad de puntos en el segundo cuadrante: $cant2")
    println("Cantidad de puntos en el tercer cuadrante: $cant3")
    println("Cantidad de puntos en el cuarto cuadrante: $cant4")
}

```

**Documentación:** Este programa en Kotlin clasifica un conjunto de puntos en los cuatro cuadrantes del plano cartesiano según sus coordenadas (x, y). Solicita al usuario ingresar la cantidad de puntos a procesar, junto con las coordenadas de cada punto, y cuenta cuántos puntos pertenecen a cada cuadrante. Al finalizar, muestra la cantidad de puntos en cada cuadrante.

#### **Proyecto 58:**

```

fun main(parametro: Array<String>) {
    var negativos = 0 // Contador para valores negativos
    var positivos = 0 // Contador para valores positivos
    var mult15 = 0 // Contador para valores múltiplos de 15
    var sumapares = 0 // Acumulador para la suma de valores pares

    // Bucle for que solicita y analiza 10 valores
    for (i in 1..10) {
        print("Ingrese valor:") // Solicita un valor al usuario
        val valor = readln().toInt() // Convierte la entrada a entero

        // Incrementa el contador de negativos si el valor es menor a 0
        if (valor < 0)
            negativos++

        // Incrementa el contador de positivos si el valor es mayor a 0
        else if (valor > 0)

```

```

        positivos++

    // Incrementa el contador de múltiplos de 15 si el valor es divisible por 15
    if (valor % 15 == 0)
        mult15++

    // Suma los valores pares si el valor es divisible por 2
    if (valor % 2 == 0)
        sumapares += valor
}

// Imprime la cantidad de valores negativos
println("Cantidad de valores negativos: $negativos")

// Imprime la cantidad de valores positivos
println("Cantidad de valores positivos: $positivos")

// Imprime la cantidad de valores múltiplos de 15
println("Cantidad de valores múltiplos de 15: $mult15")

// Imprime la suma de los valores pares
println("Suma de los valores pares: $sumapares")
}

```

**Documentación:** Este programa en Kotlin solicita al usuario ingresar diez valores enteros. Evalúa cada valor ingresado para contar cuántos son negativos, positivos y múltiplos de 15. También calcula la suma de los valores pares. Utiliza un bucle for para iterar sobre los diez valores y estructuras condicionales para realizar las clasificaciones y cálculos. Finalmente, muestra los resultados en la consola.

#### Proyecto 59:

```

fun main(parametro: Array<String>) {

    // Solicita la coordenada x del punto
    print("Ingrese coordenada x del punto:")

    val x = readLine()!!.toInt() // Convierte la entrada de x a un entero
}

```



```

// Solicita la coordenada y del punto
print("Ingrese coordenada y del punto:")

val y = readLine()!!.toInt() // Convierte la entrada de y a un entero


// Determina en qué cuadrante se encuentra el punto o si está en un eje
when {
    x > 0 && y > 0 -> println("Primer cuadrante") // Condición para el primer cuadrante
    x < 0 && y > 0 -> println("Segundo cuadrante") // Condición para el segundo cuadrante
    x < 0 && y < 0 -> println("Tercer cuadrante") // Condición para el tercer cuadrante
    x > 0 && y < 0 -> println("Cuarto cuadrante") // Condición para el cuarto cuadrante
    else -> println("El punto se encuentra en un eje") // Caso para cuando el punto está en un eje
}
}

```

**Documentación:** Este programa en Kotlin determina en cuál cuadrante del plano cartesiano se encuentra un punto dado según sus coordenadas (x, y). Utiliza la estructura when para evaluar las condiciones de pertenencia a los cuadrantes o si el punto está en alguno de los ejes. Solicita al usuario las coordenadas y muestra el resultado correspondiente en la consola.

#### Proyecto 60:

```

fun main(parametros: Array<String>) {
    // Solicita al usuario la primera nota
    print("Ingrese primer nota:")

    val nota1 = readLine()!!.toInt() // Convierte la entrada a entero


    // Solicita al usuario la segunda nota
    print("Ingrese segunda nota:")

    val nota2 = readLine()!!.toInt() // Convierte la entrada a entero


    // Solicita al usuario la tercera nota
    print("Ingrese tercer nota:")
}

```

```

val nota3 = readLine()!!.toInt() // Convierte la entrada a entero

// Calcula el promedio de las tres notas
val promedio = (nota1 + nota2 + nota3) / 3

// Clasifica al estudiante según el promedio
when {
    promedio >= 7 -> print("Promocionado") // Caso para promedio mayor o igual a 7
    promedio >= 4 -> print("Regular") // Caso para promedio entre 4 y 6
    else -> print("Reprobado") // Caso para promedio menor a 4
}
}

```

**Documentación:** Este programa en Kotlin solicita tres notas al usuario, calcula el promedio de estas y clasifica al estudiante en tres posibles estados: "Promocionado" si el promedio es mayor o igual a 7, "Regular" si está entre 4 y 6 (inclusive), y "Reprobado" si es menor a 4. La clasificación se realiza utilizando una estructura condicional when.

#### **Proyecto 61:**

```

fun main() {

    print("Ingrese un número para calcular su factorial:") // Solicita un número al usuario
    val n = readln().toInt() // Convierte la entrada a entero
    var factorial = 1 // Inicializa el resultado del factorial

    // Calcula el factorial mediante un bucle for
    for (i in 1..n) {
        factorial *= i // Multiplica el factorial por el valor actual de i
    }

    println("El factorial de $n es: $factorial") // Imprime el resultado
}

```

**Documentación:** Este programa calcula la factorial de un número entero ingresado por el usuario. Utiliza un bucle for para realizar el cálculo multiplicativo secuencial y mostrar el resultado.

**Proyecto 62:**

```
fun main() {  
    println("Números impares entre 1 y 100:")  
  
    // Bucle para recorrer los números impares  
    for (i in 1..100 step 2) {  
        println(i) // Imprime el número impar actual  
    }  
}
```

**Documentación:** Este programa imprime todos los números impares entre 1 y 100 utilizando un bucle for con un paso (step) de 2, comenzando desde 1.

**Proyecto 63:**

```
fun main() {  
    print("Ingrese el límite superior para encontrar números primos:") // Solicita el límite superior  
    val limite = readln().toInt() // Convierte la entrada a entero  
  
    println("Números primos entre 1 y $limite:")  
    for (num in 2..limite) {  
        var esPrimo = true // Inicializa la bandera para verificar si es primo  
        for (i in 2 until num) {  
            if (num % i == 0) {  
                esPrimo = false // Cambia la bandera si el número no es primo  
                break  
            }  
        }  
        if (esPrimo) println(num) // Imprime el número si es primo  
    }  
}
```

```
}
```

**Documentación:** Este programa determina qué números dentro de un rango son primos. Solicita un límite superior al usuario y evalúa cada número dentro del rango utilizando divisiones.

**Proyecto 64:**

```
fun main() {  
  
    print("Ingrese la cantidad de calificaciones:") // Solicita el número de calificaciones  
    val n = readln().toInt() // Convierte la entrada a entero  
  
    print("Ingrese el umbral mínimo:") // Solicita el valor umbral  
    val umbral = readln().toInt() // Convierte la entrada a entero  
  
  
    var suma = 0  
    var contador = 0  
  
    for (i in 1..n) {  
        print("Ingrese una calificación:") // Solicita una calificación al usuario  
        val calificacion = readln().toInt() // Convierte la entrada a entero  
        if (calificacion > umbral) {  
            suma += calificacion // Suma las calificaciones mayores al umbral  
            contador++ // Incrementa el contador  
        }  
    }  
  
    if (contador > 0) {  
        val promedio = suma / contador // Calcula el promedio  
        println("El promedio de calificaciones mayores a $umbral es: $promedio")  
    } else {  
        println("No se ingresaron calificaciones mayores a $umbral")  
    }  
}
```

**Documentación:** Este programa solicita una cantidad de calificaciones y calcula el promedio únicamente de las calificaciones que superan un valor umbral.

**Proyecto 65:**

```
fun main() {  
  
    print("Ingrese un número entero:") // Solicita un número al usuario  
  
    var numero = readln().toInt() // Convierte la entrada a entero  
  
  
    var pares = 0  
    var impares = 0  
  
    while (numero != 0) {  
        val digito = numero % 10 // Extrae el último dígito  
        if (digito % 2 == 0)  
            pares++ // Incrementa el contador de pares  
        else  
            impares++ // Incrementa el contador de impares  
  
        numero /= 10 // Elimina el último dígito  
    }  
  
    println("Dígitos pares: $pares")  
    println("Dígitos impares: $impares")  
}
```

**Documentación:** Este programa cuenta cuántos dígitos pares e impares contiene un número ingresado por el usuario.

**Proyecto 66:**

```
fun main() {  
  
    print("Ingrese un número entero:") // Solicita un número al usuario  
  
    val numero = readln().toInt() // Convierte la entrada a entero
```

```

val numeroStr = numero.toString() // Convierte el número a cadena de texto

// Compara la cadena original con su reverso
if (numeroStr == numeroStr.reversed())
    println("El número $numero es capicúa.") // Mensaje si el número es capicúa
else
    println("El número $numero no es capicúa.") // Mensaje si el número no es capicúa
}

```

**Documentación:** Este programa verifica si un número entero ingresado por el usuario es capicúa, es decir, si se lee igual de izquierda a derecha que de derecha a izquierda.

#### Proyecto 67:

```

fun main() {

    print("Ingresa un número entero N:") // Solicita el valor de N al usuario
    val n = readln().toInt() // Convierte la entrada a entero

    var sumaCuadrados = 0 // Inicializa la suma de cuadrados en 0

    // Bucle para calcular la suma de cuadrados desde 1 hasta N
    for (i in 1..n) {
        sumaCuadrados += i * i // Suma el cuadrado del número actual
    }

    println("La suma de los cuadrados de los primeros $n números es: $sumaCuadrados") // Imprime
    el resultado
}

```

**Documentación:** Este programa solicita al usuario un número entero N y calcula la suma de los cuadrados de los números desde 1 hasta N.

#### Proyecto 68:

```

fun main() {

    print("Ingresa una cadena de texto:") // Solicita una cadena de texto al usuario
}

```

```

val cadena = readln() // Lee la entrada como cadena de texto

val cantidadCaracteres = cadena.length // Calcula la cantidad de caracteres

println("La cantidad de caracteres en la cadena es: $cantidadCaracteres") // Imprime el resultado
}

```

**Documentación:** Este programa solicita al usuario una cadena de texto y cuenta cuántos caracteres tiene, incluyendo espacios.

### Proyecto 69:

```

fun main() {

    print("Ingrese la cantidad de números a ingresar:") // Solicita la cantidad de números al usuario
    val n = readln().toInt() // Convierte la entrada a entero

    var mayor = Int.MIN_VALUE // Inicializa el mayor con el valor mínimo posible
    var menor = Int.MAX_VALUE // Inicializa el menor con el valor máximo posible

    // Bucle para procesar cada número
    for (i in 1..n) {

        print("Ingrese un número:") // Solicita un número al usuario
        val numero = readln().toInt() // Convierte la entrada a entero

        if (numero > mayor)
            mayor = numero // Actualiza el mayor si el número actual es mayor
        if (numero < menor)
            menor = numero // Actualiza el menor si el número actual es menor
    }

    println("El mayor número ingresado es: $mayor") // Imprime el mayor
    println("El menor número ingresado es: $menor") // Imprime el menor
}

```

```
}
```

**Documentación:** Este programa solicita al usuario ingresar una cantidad N de números y encuentra el mayor y menor valor entre ellos.

**Proyecto 70:**

```
fun main() {  
  
    print("Ingrese un número entero para generar su tabla de multiplicar:") // Solicita un número al usuario  
  
    val numero = readln().toInt() // Convierte la entrada a entero  
  
    println("Tabla de multiplicar del $numero:")  
    for (i in 1..10) {  
        println("$numero x $i = ${numero * i}") // Calcula y muestra el resultado de la multiplicación  
    }  
}
```

**Documentación:** Este programa solicita al usuario un número entero y genera su tabla de multiplicar desde 1 hasta 10.

**Proyecto 71:**

// Función que solicita dos valores al usuario, calcula su suma y muestra el resultado

```
fun cargarSuma() {  
  
    print("Ingrese el primer valor:") // Solicita el primer valor  
    val valor1 = readLine()!!.toInt() // Convierte la entrada a entero  
  
    print("Ingrese el segundo valor:") // Solicita el segundo valor  
    val valor2 = readLine()!!.toInt() // Convierte la entrada a entero  
  
    val suma = valor1 + valor2 // Realiza la suma de los dos valores  
    println("La suma de los dos valores es: $suma") // Imprime el resultado de la suma  
}
```

// Función que imprime una línea de separación decorativa



```
fun separacion() {
    println("*****") // Línea decorativa
}
```

// Función principal que ejecuta el programa

```
fun main(parametro: Array<String>) {
    // Bucle que repite la operación de suma 5 veces
    for (i in 1..5) {
        cargarSuma() // Llama a la función para realizar la suma
        separacion() // Llama a la función para imprimir la separación
    }
}
```

**Documentación:** Este programa en Kotlin realiza la suma de dos valores ingresados por el usuario, repitiendo el proceso 5 veces. Después de cada operación de suma, se imprime una línea de separación decorativa. El programa utiliza dos funciones: `cargarSuma`, que gestiona la entrada de datos y el cálculo de la suma, y `separacion`, que imprime la línea decorativa. La ejecución del programa principal está contenida en un bucle `for` que realiza las iteraciones necesarias.

### Proyecto 72:

// Función que calcula el cuadrado de un número ingresado por el usuario

```
fun calculaCuadrado() {
    print("Ingresa un entero:") // Solicita un número entero al usuario
    val valor = readln().toInt() // Convierte la entrada a entero
    val cuadrado = valor * valor // Calcula el cuadrado del número
    println("El cuadrado es $cuadrado") // Muestra el resultado
}
```

// Función que calcula el producto de dos números ingresados por el usuario

```
fun calcularProducto() {
    print("Ingresa primer valor:") // Solicita el primer valor
    val valor1 = readln().toInt() // Convierte la entrada a entero
```

```

print("Ingrese segundo valor:") // Solicita el segundo valor

val valor2 = readln().toInt() // Convierte la entrada a entero


val producto = valor1 * valor2 // Calcula el producto de los valores

println("El producto de los valores es: $producto") // Muestra el resultado
}

```

```

// Función principal que ejecuta el programa

fun main(parametro: Array<String>) {

    calculaCuadrado() // Llama a la función para calcular el cuadrado

    calcularProducto() // Llama a la función para calcular el producto

}

```

**Documentación:** Este programa en Kotlin incluye dos funciones: `calculaCuadrado`, que calcula y muestra el cuadrado de un número entero ingresado por el usuario, y `calcularProducto`, que calcula y muestra el producto de dos valores también ingresados por el usuario. En la función principal, se llama a ambas funciones para realizar las operaciones.

### Proyecto 73:

```

// Función que encuentra el menor valor entre tres números ingresados

fun menorValor() {

    print("Ingrese primer valor:") // Solicita el primer número

    val valor1 = readln().toInt() // Convierte la entrada del primer valor a entero


    print("Ingrese segundo valor:") // Solicita el segundo número

    val valor2 = readln().toInt() // Convierte la entrada del segundo valor a entero


    print("Ingrese tercer valor:") // Solicita el tercer número

    val valor3 = readln().toInt() // Convierte la entrada del tercer valor a entero


    print("Menor de los tres:") // Mensaje previo a mostrar el menor valor

```

```
// Estructura when para determinar el menor valor entre los tres
when {
    valor1 < valor2 && valor1 < valor3 -> println(valor1) // Caso: valor1 es el menor
    valor2 < valor3 -> println(valor2) // Caso: valor2 es el menor
    else -> println(valor3) // Caso: valor3 es el menor
}
}
```

```
// Función principal que llama a la función menorValor dos veces
fun main(parametro: Array<String>) {
    menorValor() // Primera ejecución de la función menorValor
    menorValor() // Segunda ejecución de la función menorValor
}
```

**Documentación:** Este programa en Kotlin define una función menorValor que solicita tres números enteros al usuario y determina cuál es el menor entre ellos. Utiliza una estructura when para realizar la comparación de los valores. En la función principal main, se llama a menorValor dos veces para repetir el proceso.

#### Proyecto 74:

```
// Función que imprime un mensaje rodeado de líneas decorativas
fun mmostrarMensaje(mensaje: String) {
    println("*****") // Línea decorativa superior
    println(mensaje) // Imprime el mensaje recibido como parámetro
    println("*****") // Línea decorativa inferior
}
```

```
// Función que solicita dos valores al usuario y calcula su suma
fun cargarSumarPrograma70() {
```

```

print("Ingrese el primer valor:") // Solicita el primer valor

val valor1 = readLine()!!.toInt() // Convierte la entrada del primer valor a entero


print("Ingrese el segundo valor:") // Solicita el segundo valor

val valor2 = readLine()!!.toInt() // Convierte la entrada del segundo valor a entero


val suma = valor1 + valor2 // Calcula la suma de los dos valores

println("La suma de los dos valores es: $suma") // Imprime el resultado de la suma
}


// Función principal que organiza el flujo del programa
fun main(parametro: Array<String>) {

    mmostrarMensaje("El programa calcula la suma de dos valores ingresados por teclado.") //
Mensaje inicial

    cargarSumarPrograma70() // Llama a la función para calcular la suma

    mmostrarMensaje("Gracias por utilizar este programa") // Mensaje de cierre

}

```

**Documentación:** Este programa en Kotlin utiliza dos funciones: mmostrarMensaje para imprimir un mensaje rodeado de líneas decorativas, y cargarSumarPrograma70 para calcular y mostrar la suma de dos valores ingresados por el usuario. En la función principal main, se utiliza mmostrarMensaje antes y después de llamar a cargarSumarPrograma70 para indicar el propósito y cierre del programa.

#### **Proyecto 75:**

```

// Función que determina el mayor de tres valores enteros

fun mostrarMayor(v1: Int, v2: Int, v3: Int) {

    print("Mayor:") // Mensaje para indicar el resultado


// Estructura condicional para encontrar el mayor valor

    if (v1 > v2 && v1 > v3)

        println(v1) // Caso: v1 es el mayor

    else if (v2 > v3)

```

```

        println(v2) // Caso: v2 es el mayor
    else
        println(v3) // Caso: v3 es el mayor
    }

// Función principal que solicita tres valores al usuario y los pasa a la función mostrarMayor
fun main(parametro: Array<String>) {
    print("Ingrese primer valor:") // Solicita el primer valor
    val valor1 = readLine()!!.toInt() // Convierte la entrada del primer valor a entero

    print("Ingrese segundo valor:") // Solicita el segundo valor
    val valor2 = readLine()!!.toInt() // Convierte la entrada del segundo valor a entero

    print("Ingrese tercer valor:") // Solicita el tercer valor
    val valor3 = readLine()!!.toInt() // Convierte la entrada del tercer valor a entero

    mostrarMayor(valor1, valor2, valor3) // Llama a la función mostrarMayor con los valores
    ingresados
}

```

**Documentación:** Este programa en Kotlin define una función llamada `mostrarMayor`, que determina y muestra el mayor valor entre tres números pasados como parámetros. La función utiliza estructuras condicionales `if-else` para comparar los valores y mostrar el mayor. En la función principal `main`, se solicita al usuario que ingrese tres números, los cuales son pasados a `mostrarMayor` para realizar el cálculo.

#### **Proyecto 76:**

```

// Función que calcula y muestra el perímetro de un cuadrado
fun mostrarPerimetro(lado: Int) {
    val perimetro = lado * 4 // Calcula el perímetro multiplicando el lado por 4
    println("El perímetro es $perimetro") // Muestra el resultado del perímetro
}

```

```

// Función que calcula y muestra la superficie de un cuadrado
fun mostrarSuperficie(lado: Int) {
    val superficie = lado * lado // Calcula la superficie multiplicando el lado por sí mismo
    println("La superficie es $superficie") // Muestra el resultado de la superficie
}

// Función principal que organiza la ejecución del programa
fun main(parametro: Array<String>) {
    print("Ingrese el valor del lado de un cuadrado:") // Solicita al usuario el valor del lado
    val la = readLine()!!.toInt() // Convierte la entrada a entero

    print("Quiere calcular el perimetro o la superficie [ingresar texto: perimetro/superficie]:")
    // Solicita al usuario elegir la opción de cálculo
    val respuesta = readLine()!! // Lee la elección del usuario como cadena de texto

    // Evalúa la respuesta del usuario y llama a la función correspondiente
    when (respuesta) {
        "perimetro" -> mostrarPerimetro(la) // Llama a mostrarPerimetro si se eligió "perimetro"
        "superficie" -> mostrarSuperficie(la) // Llama a mostrarSuperficie si se eligió "superficie"
    }
}

```

**Documentación:** Este programa en Kotlin permite al usuario calcular el perímetro o la superficie de un cuadrado. Utiliza dos funciones: `mostrarPerimetro` para calcular el perímetro y `mostrarSuperficie` para calcular la superficie. En la función principal `main`, se solicita el lado del cuadrado y la opción del cálculo deseado (perímetro o superficie). Según la elección del usuario, se llama a la función correspondiente para realizar el cálculo y mostrar el resultado.

### Proyecto 77:

```

// Función que verifica si dos claves ingresadas son iguales o diferentes
fun verificarClaves(clave1: String, clave2: String) {

```

```

// Condicional para comparar las claves
if (clave1 == clave2)
    println("Se ingresaron las dos veces la misma clave") // Mensaje para claves iguales
else
    println("No se ingresó las dos veces con el mismo valor") // Mensaje para claves diferentes
}

// Función principal que solicita las claves al usuario y llama a verificarClaves
fun main(parametro: Array<String>) {
    print("Ingrese primer clave:") // Solicita la primera clave al usuario
    val clave1 = readln() // Lee la primera clave como cadena de texto

    print("Repita el ingreso de la misma clave:") // Solicita repetir la clave ingresada
    val clave2 = readln() // Lee la segunda clave como cadena de texto

    verificarClaves(clave1, clave2) // Llama a la función para validar las claves
}

```

**Documentación:** Este programa en Kotlin define una función llamada `verificarClaves`, que compara dos cadenas de texto para determinar si son iguales o diferentes. La función principal `main` solicita al usuario ingresar una clave dos veces y utiliza `verificarClaves` para realizar la validación, mostrando el resultado correspondiente.

#### Proyecto 78:

```

// Función que ordena tres valores en orden creciente (menor a mayor) y los imprime
fun ordenarMayorMenor(valor1: Int, valor2: Int, valor3: Int) {
    // Estructura when para ordenar los valores
    when {
        // Caso: valor1 es el menor
        valor1 < valor2 && valor1 < valor3 -> {
            if (valor2 < valor3)
                println("$valor1 $valor2 $valor3") // Orden: valor1, valor2, valor3
        }
    }
}

```

```

        else

            println("$valor1 $valor3 $valor2") // Orden: valor1, valor3, valor2
        }

        // Caso: valor2 es el menor
        valor2 < valor3 -> {

            if (valor1 < valor3)

                println("$valor2 $valor1 $valor3") // Orden: valor2, valor1, valor3
            else

                println("$valor2 $valor3 $valor1") // Orden: valor2, valor3, valor1
        }

        // Caso: valor3 es el menor
        else -> {

            if (valor1 < valor2)

                println("$valor3 $valor1 $valor2") // Orden: valor3, valor1, valor2
            else

                println("$valor3 $valor2 $valor1") // Orden: valor3, valor2, valor1
        }

    }

}

// Función principal que solicita tres valores al usuario y los pasa a ordenadrMayorMenor
fun main(parametros: Array<String>) {

    print("Ingrese primer valor:") // Solicita el primer valor al usuario
    val valor1 = readln().toInt() // Convierte la entrada del primer valor a entero


    print("Ingrese segundo valor:") // Solicita el segundo valor al usuario
    val valor2 = readln().toInt() // Convierte la entrada del segundo valor a entero


    print("Ingrese tercer valor:") // Solicita el tercer valor al usuario

```



```
val valor3 = readln().toInt() // Convierte la entrada del tercer valor a entero
```

```
ordenadrMayorMenor(valor1, valor2, valor3) // Llama a la función para ordenar y mostrar los valores
```

```
}
```

**Documentación:** Este programa en Kotlin incluye una función llamada ordenadrMayorMenor que ordena tres valores enteros pasados como parámetros en orden creciente (de menor a mayor). Utiliza una estructura when para determinar la secuencia correcta y los imprime en orden. En la función principal main, se solicita al usuario ingresar los tres valores, que luego son procesados y ordenados mediante ordenadrMayorMenor.

#### **Proyecto 79:**

```
// Función que calcula y retorna la superficie de un cuadrado
```

```
fun retornarSuperficie(lado: Int): Int {
```

```
    val sup = lado * lado // Calcula la superficie multiplicando el lado por sí mismo
```

```
    return sup // Retorna la superficie calculada
```

```
}
```

```
// Función principal que organiza la ejecución del programa
```

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

```
    print("Ingrese el valor del lado del cuadrado:") // Solicita al usuario el valor del lado
```

```
    val la = readLine()!!.toInt() // Convierte la entrada a entero
```

```
    val superficie = retornarSuperficie(la) // Llama a la función para calcular la superficie
```

```
    println("La superficie del cuadrado es $superficie") // Muestra el resultado de la superficie
```

```
}
```

**Documentación:** Este programa en Kotlin calcula la superficie de un cuadrado a partir de la longitud de su lado. Utiliza una función llamada retornarSuperficie, que recibe la longitud del lado como parámetro, calcula la superficie y la retorna. En la función principal main, se solicita al usuario ingresar la longitud del lado, se llama a retornarSuperficie con este valor, y se muestra el resultado.

#### **Proyecto 80:**

```
// Función que retorna el mayor de dos valores enteros
```

```

fun retornarMayor(v1: Int, v2: Int): Int {
    // Condicional para comparar los dos valores
    if (v1 > v2)
        return v1 // Retorna v1 si es mayor que v2
    else
        return v2 // Retorna v2 si es mayor o igual a v1
}

// Función principal que solicita dos valores al usuario y llama a retornarMayor
fun main(parametro: Array<String>) {
    print("Ingrese el primer valor:") // Solicita el primer valor al usuario
    val valor1 = readLine()!!.toInt() // Convierte la entrada del primer valor a entero

    print("Ingrese el segundo valor:") // Solicita el segundo valor al usuario
    val valor2 = readLine()!!.toInt() // Convierte la entrada del segundo valor a entero

    // Llama a la función retornarMayor y muestra el mayor valor
    println("El mayor entre $valor1 y $valor2 es ${retornarMayor(valor1, valor2)}")
}

```

**Documentación:** Este programa en Kotlin define una función llamada `retornarMayor`, que recibe dos valores enteros como parámetros y retorna el mayor de los dos. En la función principal `main`, se solicita al usuario que ingrese dos valores, se llama a la función `retornarMayor` y se imprime el resultado, indicando cuál de los dos valores es mayor.

#### Proyecto 81:

```

// Función que calcula y retorna la longitud de una cadena de texto
fun largo(nombre: String): Int {
    return nombre.length // Retorna la cantidad de caracteres de la cadena
}

// Función principal que organiza la ejecución del programa

```

```

fun main(parametro: Array<String>) {

    print("Ingrese un nombre:") // Solicita el primer nombre al usuario

    val nombre1 = readLine()!! // Lee el primer nombre como cadena de texto


    print("Ingrese otro nombre:") // Solicita el segundo nombre al usuario

    val nombre2 = readLine()!! // Lee el segundo nombre como cadena de texto


    // Compara las longitudes de los nombres y muestra el resultado

    if (largo(nombre1) == largo(nombre2))

        print("Los nombres: $nombre1 y $nombre2 tienen la misma cantidad de caracteres") // Mensaje
        si tienen la misma longitud

    else if (largo(nombre1) > largo(nombre2))

        print("$nombre1 es más largo") // Mensaje si el primer nombre es más largo

    else

        print("$nombre2 es más largo") // Mensaje si el segundo nombre es más largo

}

```

**Documentación:** Este programa en Kotlin permite comparar la longitud de dos nombres ingresados por el usuario. Utiliza una función llamada largo que retorna la cantidad de caracteres de una cadena dada. En la función principal main, se determina si ambos nombres tienen la misma longitud o cuál de ellos es más largo, mostrando el resultado correspondiente.

#### **Proyecto 82:**

```

// Función que calcula y retorna el promedio de tres valores enteros

fun retornarPromedio(v1: Int, v2: Int, v3: Int): Int {

    val promedio = (v1 + v2 + v3) / 3 // Calcula el promedio sumando los valores y dividiendo entre 3

    return promedio // Retorna el promedio calculado

}

```

// Función principal que organiza la ejecución del programa

```

fun main(parametro: Array<String>) {

    print("Ingrese primer valor:") // Solicita el primer valor al usuario

```

```

val valor1 = readln().toInt() // Convierte la entrada del primer valor a entero

print("Ingrese segundo valor:") // Solicita el segundo valor al usuario
val valor2 = readln().toInt() // Convierte la entrada del segundo valor a entero

print("Ingrese tercer valor:") // Solicita el tercer valor al usuario
val valor3 = readln().toInt() // Convierte la entrada del tercer valor a entero

// Llama a la función retornarPromedio y muestra el resultado

println("Valor promedio de los tres números ingresados es ${retornarPromedio(valor1, valor2,
valor3)}")
}

```

**Documentación:** Este programa en Kotlin calcula el promedio de tres valores enteros ingresados por el usuario. Define una función llamada `retornarPromedio`, que realiza la operación del promedio y lo retorna. En la función principal `main`, se solicitan los tres valores al usuario, se llama a `retornarPromedio`, y se muestra el resultado en la consola.

### Proyecto 83:

```

// Función que calcula y retorna el perímetro de un cuadrado
fun retornarPerimetro(lado: Int): Int {
    val perimetro = lado * 4 // Calcula el perímetro multiplicando el lado por 4
    return perimetro // Retorna el perímetro calculado
}

// Función principal que organiza la ejecución del programa
fun main(parametro: Array<String>) {
    print("Ingrese el lado del cuadrado:") // Solicita al usuario el valor del lado
    val lado = readln().toInt() // Convierte la entrada a entero

    // Llama a la función retornarPerimetro y muestra el resultado
    print("El perimetro es: ${retornarPerimetro(lado)}")
}

```

```
}
```

**Documentación:** Este programa en Kotlin calcula el perímetro de un cuadrado utilizando una función llamada `retornarPerimetro`. Esta función recibe la longitud de un lado como parámetro, multiplica dicho valor por 4, y retorna el resultado. En la función principal `main`, se solicita al usuario ingresar el valor del lado del cuadrado, se llama a `retornarPerimetro` con este valor, y se muestra el perímetro calculado en la consola.

#### Proyecto 84:

```
// Función que calcula y retorna la superficie de un rectángulo
```

```
fun retornarSuperficie(lado1: Int, lado2: Int): Int {  
    return lado1 * lado2 // Retorna la superficie multiplicando los dos lados  
}
```

```
// Función principal que solicita las dimensiones de dos rectángulos y compara sus superficies
```

```
fun main(parametro: Array<String>) {  
    println("Primer rectángulo") // Indica que se solicitarán datos del primer rectángulo  
    print("Ingresa lado menor del rectángulo:") // Solicita el lado menor del primer rectángulo  
    val lado1 = readln().toInt() // Convierte la entrada a entero  
  
    print("Ingresa lado mayor del rectángulo:") // Solicita el lado mayor del primer rectángulo  
    val lado2 = readln().toInt() // Convierte la entrada a entero  
  
    println("Segundo rectángulo") // Indica que se solicitarán datos del segundo rectángulo  
    print("Ingresa lado menor del rectángulo:") // Solicita el lado menor del segundo rectángulo  
    val lado3 = readln().toInt() // Convierte la entrada a entero  
  
    print("Ingresa lado mayor del rectángulo:") // Solicita el lado mayor del segundo rectángulo  
    val lado4 = readln().toInt() // Convierte la entrada a entero  
  
    // Compara las superficies de los dos rectángulos  
    if (retornarSuperficie(lado1, lado2) == retornarSuperficie(lado3, lado4)) {
```

```

        print("Los dos rectángulos tienen la misma superficie") // Mensaje si las superficies son iguales
    } else if (retornarSuperficie(lado1, lado2) > retornarSuperficie(lado3, lado4)) {
        print("El primer rectángulo tiene una superficie mayor") // Mensaje si el primer rectángulo tiene
mayor superficie
    } else {
        print("El segundo rectángulo tiene una superficie mayor") // Mensaje si el segundo rectángulo
tiene mayor superficie
    }
}
}

```

**Documentación:** Este programa en Kotlin calcula y compara las superficies de dos rectángulos. Define una función `retornarSuperficie` que recibe las dimensiones de un rectángulo (lado menor y lado mayor) como parámetros y retorna su superficie. La función principal `main` solicita al usuario ingresar las dimensiones de dos rectángulos y, utilizando `retornarSuperficie`, determina si ambos tienen la misma superficie o cuál es mayor.

#### Proyecto 85:

```

// Función que calcula y retorna la superficie de un cuadrado
fun calcularSuperficieCuadrado(lado: Int) = lado * lado // Retorna el lado multiplicado por sí mismo

// Función principal que organiza la ejecución del programa
fun main(parametro: Array<String>) {
    print("Ingrese el valor del lado del cuadrado:") // Solicita al usuario el valor del lado
    val la = readLine()!!.toInt() // Convierte la entrada a entero

    // Llama a la función calcularSuperficieCuadrado y muestra el resultado
    println("La superficie del cuadrado es ${calcularSuperficieCuadrado(la)}")
}

```

**Documentación:** Este programa en Kotlin calcula la superficie de un cuadrado utilizando una función llamada `calcularSuperficieCuadrado`. La función utiliza una expresión simplificada (`=`) para retornar directamente el resultado de la multiplicación del lado por sí mismo. En la función principal `main`, se solicita al usuario el valor del lado del cuadrado, se llama a la función para calcular la superficie y se imprime el resultado.

#### Proyecto 86:

```
// Función que retorna el mayor de dos valores enteros utilizando una expresión simplificada

fun devolverMayor(v1: Int, v2: Int) = if (v1 > v2) v1 else v2 // Retorna v1 si es mayor que v2; de lo contrario, retorna v2


// Función principal que organiza la ejecución del programa

fun main(parametro: Array<String>) {

    print("Ingrese el primer valor:") // Solicita al usuario el primer valor

    val valor1 = readLine()!!.toInt() // Convierte la entrada del primer valor a entero


    print("Ingrese el segundo valor:") // Solicita al usuario el segundo valor

    val valor2 = readLine()!!.toInt() // Convierte la entrada del segundo valor a entero


    // Llama a la función devolverMayor y muestra el resultado

    println("El mayor entre $valor1 y $valor2 es ${devolverMayor(valor1, valor2)}")

}
```

**Documentación:** Este programa en Kotlin encuentra el mayor de dos valores enteros ingresados por el usuario. Utiliza una función llamada `devolverMayor`, que utiliza una expresión simplificada (`=`) para retornar directamente el valor mayor entre los dos números. En la función principal `main`, se solicita al usuario ingresar los dos valores, se llama a la función y se muestra el resultado en la consola.

### Proyecto 87:

```
// Función que convierte números enteros en su representación en español

fun convertirCastelano(valor: Int) = when (valor) {

    1 -> "uno" // Retorna "uno" si el valor es 1

    2 -> "dos" // Retorna "dos" si el valor es 2

    3 -> "tres" // Retorna "tres" si el valor es 3

    4 -> "cuatro" // Retorna "cuatro" si el valor es 4

    5 -> "cinco" // Retorna "cinco" si el valor es 5

    else -> "error" // Retorna "error" si el valor no está en el rango de 1 a 5

}
```

```
// Función principal que prueba la función convertirCastelano con valores del 1 al 6
```

```
fun main(parametro: Array<String>) {  
    for (i in 1..6) { // Itera del 1 al 6  
        println(convertirCastelano(i)) // Llama a la función y muestra el resultado  
    }  
}
```

**Documentación:** Este programa en Kotlin incluye una función llamada `convertirCastelano`, que convierte números enteros en su representación textual en español para valores entre 1 y 5. Si el número no está dentro de este rango, retorna el texto "error". La función principal `main` utiliza un bucle `for` para probar la función con los valores del 1 al 6, mostrando los resultados correspondientes en la consola.

#### Proyecto 88:

```
// Función que calcula y retorna el promedio de tres valores enteros utilizando una expresión simplificada
```

```
fun devolverPromedio(v1: Int, v2: Int, v3: Int) = (v1 + v2 + v3) / 3 // Retorna el promedio sumando los valores y dividiendo entre 3
```

```
// Función principal que organiza la ejecución del programa
```

```
fun main(parametro: Array<String>) {  
    print("Ingrese primer valor:") // Solicita el primer valor al usuario  
    val valor1 = readln().toInt() // Convierte la entrada del primer valor a entero  
  
    print("Ingrese segundo valor:") // Solicita el segundo valor al usuario  
    val valor2 = readln().toInt() // Convierte la entrada del segundo valor a entero  
  
    print("Ingrese tercer valor:") // Solicita el tercer valor al usuario  
    val valor3 = readln().toInt() // Convierte la entrada del tercer valor a entero  
  
    // Llama a la función devolverPromedio y muestra el resultado  
    println("Valor promedio de los tres números ingresados es ${devolverPromedio(valor1, valor2, valor3)}")
```



```
}
```

**Documentación:** Este programa en Kotlin calcula el promedio de tres valores enteros ingresados por el usuario. Utiliza una función simplificada llamada `devolverPromedio`, que directamente retorna el promedio utilizando una fórmula compacta. En la función principal `main`, se solicitan los tres valores al usuario, se llama a la función y se muestra el resultado en la consola.

#### Proyecto 89:

```
// Función que calcula y retorna el perímetro de un cuadrado utilizando una expresión simplificada
```

```
fun devolverPerimetro(lado: Int) = lado * 4 // Retorna el perímetro multiplicando el lado por 4
```

```
// Función principal que organiza la ejecución del programa
```

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

```
    print("Ingrese el lado del cuadrado:") // Solicita al usuario el valor del lado
```

```
    val lado = readln().toInt() // Convierte la entrada a entero
```

```
    // Llama a la función devolverPerimetro y muestra el resultado
```

```
    print("El perímetro es: ${devolverPerimetro(lado)}")
```

```
}
```

**Documentación:** Este programa en Kotlin calcula el perímetro de un cuadrado. Utiliza una función simplificada llamada `devolverPerimetro`, que directamente retorna el perímetro multiplicando el lado por 4. En la función principal `main`, se solicita al usuario ingresar la longitud del lado del cuadrado, se llama a la función y se muestra el resultado.

#### Proyecto 90:

```
// Función que calcula y retorna la superficie de un rectángulo
```

```
fun devolverSuperficie(lado1: Int, lado2: Int): Int = lado1 * lado2 // Retorna la superficie multiplicando lado menor por lado mayor
```

```
// Función principal que organiza la ejecución del programa
```

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

```
    println("Primer rectángulo") // Indica el inicio del ingreso de datos para el primer rectángulo
```

```
    print("Ingrese lado menor del rectángulo:") // Solicita el lado menor del primer rectángulo
```

```
    val lado1 = readln().toInt() // Convierte la entrada a entero
```

```

print("Ingrese lado mayor del rectángulo:") // Solicita el lado mayor del primer rectángulo
val lado2 = readln().toInt() // Convierte la entrada a entero

println("Segundo rectángulo") // Indica el inicio del ingreso de datos para el segundo rectángulo
print("Ingrese lado menor del rectángulo:") // Solicita el lado menor del segundo rectángulo
val lado3 = readln().toInt() // Convierte la entrada a entero

print("Ingrese lado mayor del rectángulo:") // Solicita el lado mayor del segundo rectángulo
val lado4 = readln().toInt() // Convierte la entrada a entero

// Compara las superficies de los dos rectángulos utilizando la función devolverSuperficie
if (devolverSuperficie(lado1, lado2) == devolverSuperficie(lado3, lado4)) {
    print("Los dos rectángulos tienen la misma superficie") // Mensaje si las superficies son iguales
} else if (devolverSuperficie(lado1, lado2) > devolverSuperficie(lado3, lado4)) {
    print("El primer rectángulo tiene una superficie mayor") // Mensaje si el primer rectángulo tiene
    mayor superficie
} else {
    print("El segundo rectángulo tiene una superficie mayor") // Mensaje si el segundo rectángulo
    tiene mayor superficie
}
}

```

**Documentación:** Este programa en Kotlin calcula y compara las superficies de dos rectángulos. Utiliza una función llamada `devolverSuperficie`, que emplea una expresión simplificada para retornar la superficie multiplicando los lados menor y mayor de un rectángulo. En la función principal `main`, se solicitan las dimensiones de dos rectángulos al usuario y se determina si tienen la misma superficie o cuál de ellos tiene una superficie mayor.

#### **Proyecto 91:**

```

// Función que calcula y retorna la longitud de un nombre

fun largoNom(nombre: String) = nombre.length // Retorna la cantidad de caracteres del nombre

```

```
// Función principal que compara la longitud de dos nombres ingresados por el usuario
fun main(parametro: Array<String>) {
    print("Ingrese un nombre:") // Solicita al usuario el primer nombre
    val nombre1 = readln() // Lee el primer nombre como cadena de texto

    print("Ingrese otro nombre:") // Solicita al usuario el segundo nombre
    val nombre2 = readln() // Lee el segundo nombre como cadena de texto

    // Compara las longitudes de los nombres utilizando funciones
    if (largoNom(nombre1) == largoNom(nombre2))
        print("Los nombres: $nombre1 y $nombre2 tienen la misma cantidad de caracteres") // Mensaje
        si tienen igual longitud
    else if (largoNom(nombre1) > largoNom(nombre2))
        print("$nombre1 es más largo") // Mensaje si el primer nombre es más largo
    else
        print("$nombre2 es más largo") // Mensaje si el segundo nombre es más largo
}
```

**Documentación:** Este programa en Kotlin compara la longitud de dos nombres ingresados por el usuario. Utiliza una función simplificada llamada `largoNom`, que retorna directamente la cantidad de caracteres de un nombre. En la función principal `main`, se solicita a los usuarios que ingresen dos nombres y luego se evalúa si tienen la misma longitud o cuál es más largo. Cabe destacar que se utiliza otra función llamada `largo` que realiza la misma operación pero no se encuentra definida en este código, por lo que se debe agregar o reemplazar por `largoNom`.

#### Proyecto 92:

```
// Función que imprime un título seguido de una línea decorativa hecha con caracteres
fun tituloSubrayado(titulo: String, caracter: String = "*") { // El parámetro 'caracter' tiene un valor
    por defecto: '*'
    println(titulo) // Imprime el título
    for (i in 1..titulo.length) // Itera según la longitud del título
        print(caracter) // Imprime el carácter especificado
    println() // Salta a la siguiente línea
}
```

```
}
```

```
// Función principal que llama a la función tituloSubrayado con distintos parámetros
```

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

```
    tituloSubrayado("Sistema de Administracion") // Usa el carácter por defecto (*)
```

```
    tituloSubrayado("Ventas", "-") // Usa el carácter personalizado ('-')
```

```
}
```

**Documentación:** Este programa en Kotlin define una función llamada tituloSubrayado, que imprime un título seguido de una línea decorativa hecha con un carácter específico (por defecto, el carácter es \*). La función principal main llama a tituloSubrayado dos veces para demostrar su funcionalidad, utilizando el valor por defecto y un carácter personalizado.

### Proyecto 93:

```
// Función que calcula y retorna la suma de hasta cinco valores enteros
```

```
fun sumar(v1: Int, v2: Int, v3: Int = 0, v4: Int = 0, v5: Int = 0) = v1 + v2 + v3 + v4 + v5
```

```
// Los parámetros v3, v4 y v5 tienen valores por defecto de 0
```

```
// Función principal que demuestra el uso de la función sumar
```

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

```
    println("La suma de 5 + 6 es ${sumar(5, 6)}") // Llama a la función sumar con 2 argumentos
```

```
    println("La suma de 1 + 2 + 3 es ${sumar(1, 2, 3)}") // Llama a la función sumar con 3 argumentos
```

```
    print("La suma de 1 + 2 + 3 + 4 + 5 es ${sumar(1, 2, 3, 4, 5)}") // Llama a la función sumar con 5 argumentos
```

```
}
```

### Documentación:

Este programa en Kotlin define una función llamada sumar, que calcula la suma de hasta cinco valores enteros. La función utiliza parámetros con valores por defecto (inicializados en 0) para permitir la suma de 2, 3, 4 o 5 números. En la función principal main, se presentan ejemplos de cómo invocar la función sumar con diferentes cantidades de parámetros, y se imprimen los resultados.

### Proyecto 94:

```
fun convertirCelsiusAFahrenheit(celsius: Double): Double {
```

```
    return (celsius * 9 / 5) + 32 // Convierte Celsius a Fahrenheit
```

```
}
```

```
fun main() {  
    print("Ingrese la temperatura en Celsius:")  
    val celsius = readln().toDouble() // Solicita la temperatura en Celsius  
    println("La temperatura en Fahrenheit es: ${convertirCelsiusAFahrenheit(celsius)}")  
}
```

**Documentación:** Este programa convierte una temperatura dada en Celsius a Fahrenheit usando la fórmula estándar.

#### **Proyecto 95:**

```
fun esPar(numero: Int): Boolean {  
    return numero % 2 == 0 // Devuelve true si el número es divisible entre 2  
}
```

```
fun main() {  
    print("Ingrese un número:")  
    val numero = readln().toInt()  
    if (esPar(numero))  
        println("El número $numero es par")  
    else  
        println("El número $numero es impar")  
}
```

**Documentación:** Determina si un número ingresado por el usuario es par o impar.

#### **Proyecto 96:**

```
fun calcularRaizCuadrada(numero: Double): Double {  
    return Math.sqrt(numero) // Calcula la raíz cuadrada  
}
```

```
fun main() {
```

```

print("Ingrese un número:")

val numero = readln().toDouble()

println("La raíz cuadrada de $numero es ${calcularRaizCuadrada(numero)}")
}

```

**Documentación:** Calcula la raíz cuadrada de un número usando la función Math.sqrt.

#### Proyecto 97:

```

fun generarNumeroAleatorio(min: Int, max: Int): Int {
    return (min..max).random() // Genera un número aleatorio en el rango
}

```

```

fun main() {
    print("Ingrese el valor mínimo:")
    val min = readln().toInt()
    print("Ingrese el valor máximo:")
    val max = readln().toInt()
    println("Número aleatorio generado: ${generarNumeroAleatorio(min, max)}")
}

```

**Documentación:** Genera un número aleatorio dentro de un rango definido por el usuario.

#### Proyecto 98:

```

fun contienePalabra(cadena: String, palabra: String): Boolean {
    return cadena.contains(palabra) // Verifica si la cadena contiene la palabra
}

```

```

fun main() {
    print("Ingrese una cadena:")
    val cadena = readln()
    print("Ingrese una palabra para buscar:")
    val palabra = readln()
    if (contienePalabra(cadena, palabra))

```

```

        println("La cadena contiene la palabra $palabra")
    else
        println("La cadena no contiene la palabra $palabra")
    }

```

**Documentación:** Verifica si una cadena ingresada contiene una palabra específica.

**Proyecto 99:**

```

fun contarVocales(cadena: String): Int {
    val vocales = "aeiouAEIOU" // Definimos las vocales
    return cadena.count { it in vocales } // Cuenta las letras que son vocales
}

```

```

fun main() {
    print("Ingrese una cadena de texto:")
    val cadena = readln()
    println("La cantidad de vocales en la cadena es: ${contarVocales(cadena)}")
}

```

**Documentación:** Cuenta la cantidad de vocales en una cadena ingresada por el usuario.

**Proyecto 100:**

```

fun calcularFactorial(numero: Int): Long {
    var factorial = 1L
    for (i in 1..numero) {
        factorial *= i // Multiplica los números consecutivamente
    }
    return factorial
}

```

```

fun main() {
    print("Ingrese un número entero positivo:")
    val numero = readln().toInt()
}

```

```
println("El factorial de $numero es: ${calcularFactorial(numero)}")
}
```

**Documentación:** Calcula el factorial de un número entero positivo ingresado por el usuario.

**Proyecto 101:**

```
fun palabraMasLarga(cadena: String): String {
    return cadena.split(" ").maxByOrNull { it.length } ?: "" // Encuentra la palabra más larga
}
```

```
fun main() {
    print("Ingrese una cadena de texto:")
    val cadena = readln()
    println("La palabra más larga es: '${palabraMasLarga(cadena)}'")
}
```

**Documentación:** Determina cuál es la palabra más larga en una cadena ingresada por el usuario.

**Proyecto 102:**

```
fun esPrimo(numero: Int): Boolean {
    if (numero < 2) return false
    for (i in 2..numero / 2) {
        if (numero % i == 0) return false // Si es divisible, no es primo
    }
    return true
}
```

```
fun main() {
    print("Ingrese un número:")
    val numero = readln().toInt()
    if (esPrimo(numero))
        println("El número $numero es primo")
    else
```



```
        println("El número $numero no es primo")
    }
}
```

**Documentación:** Determina si un número ingresado por el usuario es primo.

**Proyecto 103:**

```
fun invertirCadena(cadena: String): String {
    return cadena.reversed() // Invierte la cadena
}
```

```
fun main() {
    print("Ingrese una cadena:")
    val cadena = readln()
    println("La cadena invertida es: ${invertirCadena(cadena)}")
}
```

**Documentación:** Invierte el orden de los caracteres de una cadena ingresada por el usuario.

**Proyecto 104:**

```
fun promedioLista(lista: List<Int>): Double {
    return lista.average() // Calcula el promedio de la lista
}
```

```
fun main() {
    print("Ingrese números separados por espacios:")
    val numeros = readln().split(" ").map { it.toInt() }
    println("El promedio de los números es: ${promedioLista(numeros)}")
}
```

**Documentación:** Calcula el promedio de una lista de números enteros ingresados por el usuario.

**Proyecto 105:**

```
fun esPalindromo(palabra: String): Boolean {
    return palabra.equals(palabra.reversed(), ignoreCase = true) // Compara la palabra con su reverso
}
```

```

fun main() {
    print("Ingrese una palabra:")
    val palabra = readln()
    if (esPalindromo(palabra))
        println("La palabra '$palabra' es un palíndromo")
    else
        println("La palabra '$palabra' no es un palíndromo")
}

```

**Documentación:** Determina si una palabra ingresada por el usuario es un palíndromo.

**Proyecto 106:**

```

fun calcularMCD(a: Int, b: Int): Int {
    return if (b == 0) a else calcularMCD(b, a % b) // Algoritmo recursivo
}

```

```

fun main() {
    print("Ingrese el primer número:")
    val a = readln().toInt()
    print("Ingrese el segundo número:")
    val b = readln().toInt()
    println("El MCD de $a y $b es: ${calcularMCD(a, b)}")
}

```

**Documentación:** Calcula el MCD de dos números usando el algoritmo de Euclides.

**Proyecto 107:**

```

fun calcularAreaTriangulo(base: Double, altura: Double): Double {
    return (base * altura) / 2 // Fórmula del área
}

```

```

fun main() {

```

```

print("Ingrese la base del triángulo:")

val base = readln().toDouble()

print("Ingrese la altura del triángulo:")

val altura = readln().toDouble()

println("El área del triángulo es: ${calcularAreaTriangulo(base, altura)}")
}

```

**Documentación:** Calcula el área de un triángulo dados su base y altura.

**Proyecto 108:**

```

fun contarPalabras(cadena: String): Int {

    return cadena.split(" ").filter { it.isNotEmpty() }.size // Cuenta las palabras no vacías
}

fun main() {

    print("Ingrese una cadena de texto:")

    val cadena = readln()

    println("La cantidad de palabras en la cadena es: ${contarPalabras(cadena)}")
}

```

**Documentación:** Cuenta cuántas palabras contiene una cadena ingresada.

**Proyecto 109:**

```

fun tablaMultiplicar(numero: Int) {

    for (i in 1..10) {

        println("$numero x $i = ${numero * i}") // Genera cada línea de la tabla
    }

}

fun main() {

    print("Ingrese un número:")

    val numero = readln().toInt()

    tablaMultiplicar(numero)
}

```

```
}
```

**Documentación:** Genera la tabla de multiplicar de un número.

**Proyecto 110:**

```
fun esPerfecto(numero: Int): Boolean {  
    val sumaDivisores = (1 until numero).filter { numero % it == 0 }.sum()  
    return sumaDivisores == numero  
}
```

```
fun main() {  
    print("Ingrese
```

**Documentación:** Determina si un número es perfecto, es decir, si la suma de sus divisores propios es igual al número.

**Proyecto 111:**

// Definición de la clase Alumno

```
class Alumno {  
    var nombre: String = "" // Propiedad para almacenar el nombre del alumno  
    var nota: Int = 0 // Propiedad para almacenar la nota del alumno  
  
    // Método para inicializar las propiedades ingresadas por el usuario  
    fun inicializar() {  
        print("Ingrese el nombre del alumno:") // Solicita el nombre del alumno  
        nombre = readln().toString() // Lee el nombre como cadena de texto  
        print("Ingrese su nota:") // Solicita la nota del alumno  
        nota = readln().toInt() // Convierte la entrada a entero y la almacena en 'nota'  
    }  
}
```

// Método para imprimir los datos del alumno

```
fun imprimir() {  
    println("Alumno: $nombre tiene una nota de $nota") // Imprime el nombre y la nota
```

```

    }

    // Método para mostrar el estado del alumno (regular o no regular)
    fun mostrarEstado() {
        if (nota >= 4) // Verifica si la nota es mayor o igual a 4
            println("$nombre se encuentra en estado REGULAR") // Mensaje para estado regular
        else
            println("$nombre no está REGULAR") // Mensaje para estado no regular
        }
    }
}

// Función principal que instancia dos objetos de la clase Alumno y los gestiona
fun main(parametros: Array<String>) {
    val alumno1 = Alumno() // Crea el primer objeto Alumno
    alumno1.inicializar() // Inicializa las propiedades del primer alumno
    alumno1.imprimir() // Imprime la información del primer alumno
    alumno1.mostrarEstado() // Muestra el estado del primer alumno

    val alumno2 = Alumno() // Crea el segundo objeto Alumno
    alumno2.inicializar() // Inicializa las propiedades del segundo alumno
    alumno2.imprimir() // Imprime la información del segundo alumno
    alumno2.mostrarEstado() // Muestra el estado del segundo alumno
}

```

**Documentación:** Este programa define una clase Alumno con propiedades para almacenar el nombre y la nota de un alumno. La clase incluye métodos para inicializar los datos ingresados por el usuario, imprimir la información del alumno y verificar si el alumno está en estado regular (nota mayor o igual a 4). El programa principal crea dos objetos de la clase Alumno, permite al usuario ingresar los datos correspondientes y muestra los resultados.

**Proyecto 112:**

// Clase Persona2 con constructor que inicializa las propiedades

```

class Persona2 constructor(nombre: String, edad: Int) {

    var nombre: String = nombre // Almacena el nombre de la persona

    var edad: Int = edad // Almacena la edad de la persona


    // Método para imprimir los datos de la persona
    fun imprimir() {
        println("Nombre: $nombre y tiene una edad de $edad") // Muestra el nombre y la edad
    }


    // Método para verificar si la persona es mayor de edad
    fun esMayorEdad() {
        if (edad >= 18) // Condición para comprobar la mayoría de edad
            println("Es mayor de edad $nombre") // Mensaje si es mayor de edad
        else
            println("No es mayor de edad $nombre") // Mensaje si no es mayor de edad
    }
}


// Función principal que prueba la funcionalidad de la clase Persona2
fun main(parametro: Array<String>) {
    val persona1 = Persona2("Juan", 12) // Crea un objeto de tipo Persona2 con valores iniciales
    persona1.imprimir() // Llama al método imprimir
    persona1.esMayorEdad() // Llama al método esMayorEdad
}

```

**Documentación:** Este programa define una clase Persona2 con un constructor que inicializa las propiedades nombre y edad. La clase incluye métodos para imprimir los datos de una persona y para verificar si es mayor de edad. En la función main, se crea un objeto de tipo Persona2 con valores iniciales, y se prueban sus métodos.

### Proyecto 113:

// Definición de la clase Triangulos con propiedades para almacenar los lados del triángulo

```

class Triangulos (var lado1: Int, var lado2: Int, var lado3: Int) {

    // Método para encontrar el lado más largo del triángulo
    fun ladoMayor() {
        print("Lado mayor:") // Mensaje inicial
        when {
            lado1 > lado2 && lado1 > lado3 -> println(lado1) // Caso: lado1 es el mayor
            lado2 > lado3 -> println(lado2) // Caso: lado2 es el mayor
            else -> println(lado3) // Caso: lado3 es el mayor
        }
    }

    // Método para verificar si el triángulo es equilátero
    fun esEquilatero() {
        if (lado1 == lado2 && lado1 == lado3) // Condición para equilátero: los tres lados iguales
            print("Es un triángulo equilátero") // Mensaje para triángulo equilátero
        else
            print("No es un triángulo equilátero") // Mensaje para triángulo no equilátero
    }
}

// Función principal que prueba los métodos de la clase Triangulos
fun main(parametro: Array<String>) {
    val triangulo1 = Triangulos(12, 45, 24) // Crea un objeto de tipo Triangulos con valores de ejemplo
    triangulo1.ladoMayor() // Llama al método ladoMayor
    triangulo1.esEquilatero() // Llama al método esEquilatero
}

```

**Documentación:** Este programa define una clase Triangulos que permite trabajar con los lados de un triángulo, determinando cuál es el lado más largo y verificando si el triángulo es equilátero. La

función principal main instancia un objeto de la clase Triangulos con valores de ejemplo y utiliza los métodos proporcionados para mostrar las características del triángulo.

**Proyecto 114:**

// Clase Triangulo2 que representa las propiedades y métodos de un triángulo

```
class Triangulo2 (var lado1: Int, var lado2: Int, var lado3: Int) {

    // Constructor por defecto que permite ingresar los lados de forma interactiva
    constructor() : this(0, 0, 0) { // Inicializa los lados en 0

        print("Ingrese primer lado:") // Solicita el primer lado
        lado1 = readLine()!!.toInt() // Lee el primer lado como entero

        print("Ingrese segundo lado:") // Solicita el segundo lado
        lado2 = readLine()!!.toInt() // Lee el segundo lado como entero

        print("Ingrese tercer lado:") // Solicita el tercer lado
        lado3 = readLine()!!.toInt() // Lee el tercer lado como entero
    }

    // Método para encontrar y mostrar el lado más largo del triángulo
    fun ladoMayor() {
        print("Lado mayor:") // Mensaje inicial
        when { // Estructura when para determinar el lado mayor
            lado1 > lado2 && lado1 > lado3 -> println(lado1) // Caso: lado1 es el mayor
            lado2 > lado3 -> println(lado2) // Caso: lado2 es el mayor
            else -> println(lado3) // Caso: lado3 es el mayor
        }
    }

    // Método para verificar si el triángulo es equilátero
```



```

fun esEquilatero() {
    if (lado1 == lado2 && lado1 == lado3) // Verifica si los tres lados son iguales
        println("Es un triángulo equilátero") // Mensaje si es equilátero
    else
        println("No es un triángulo equilátero") // Mensaje si no es equilátero
    }
}

// Función principal que prueba la funcionalidad de la clase Triangulo2
fun main(parametro: Array<String>) {
    // Crea un objeto Triangulo2 usando el constructor por defecto (interactivo)
    val triangulo1 = Triangulo2()
    triangulo1.ladoMayor() // Llama al método para mostrar el lado mayor
    triangulo1.esEquilatero() // Llama al método para verificar si es equilátero

    // Crea un segundo objeto Triangulo2 con valores establecidos
    val triangulo2 = Triangulo2(6, 6, 6)
    triangulo2.ladoMayor() // Llama al método para mostrar el lado mayor
    triangulo2.esEquilatero() // Llama al método para verificar si es equilátero
}

```

**Documentación:** Este programa define una clase llamada Triangulo2, que permite trabajar con triángulos y analizar sus características. La clase incluye un constructor por defecto para inicializar los lados interactivamente, métodos para determinar el lado mayor y para verificar si el triángulo es equilátero. En la función principal main, se crean y gestionan dos objetos de la clase Triangulo2.

#### **Proyecto 115:**

```

// Clase Alumno3 con propiedades para almacenar el nombre y la nota del alumno
class Alumno3(val nombre: String, val nota: Int) {

    // Método para imprimir los datos del alumno
    fun imprimir() {

```

```

        println("Alumno: $nombre tiene una nota de $nota") // Imprime el nombre y la nota del alumno
    }

    // Método para mostrar si el alumno está en estado regular
    fun mostrarEstado() {
        if (nota >= 4) // Verifica si la nota es mayor o igual a 4
            println("$nombre se encuentra en estado REGULAR") // Mensaje si la nota es suficiente
        else
            println("$nombre no está REGULAR") // Mensaje si la nota es insuficiente
        }
    }

    // Función principal que instancia y gestiona dos objetos de la clase Alumno3
    fun main(parametros: Array<String>) {
        val alumno1 = Alumno3("Ana", 7) // Crea un objeto de tipo Alumno3 con nombre y nota
        //predefinidos

        alumno1.imprimir() // Llama al método imprimir del objeto alumno1

        alumno1.mostrarEstado() // Llama al método mostrarEstado del objeto alumno1

        val alumno2 = Alumno3("Carlos", 2) // Crea un segundo objeto de tipo Alumno3 con diferentes
        //valores

        alumno2.imprimir() // Llama al método imprimir del objeto alumno2

        alumno2.mostrarEstado() // Llama al método mostrarEstado del objeto alumno2
    }

```

**Documentación:** Este programa define una clase llamada Alumno3, que almacena las propiedades nombre y nota de un alumno. El constructor de la clase recibe ambos valores como parámetros al momento de crear un objeto. La clase incluye métodos para imprimir los datos del alumno y para mostrar si el alumno está "regular" (nota mayor o igual a 4). En la función principal main, se crean dos objetos de la clase Alumno3 con datos predefinidos y se ejecutan sus métodos.

#### **Proyecto 116:**

```

// Clase Punto que representa un punto en el plano cartesiano

```

```

class Punto(val x: Int, val y: Int) {

    // Método que determina el cuadrante o si el punto está en un eje
    fun retornarCuadrante() = when {

        x > 0 && y > 0 -> "Primer cuadrante" // Caso: x > 0, y > 0
        x < 0 && y > 0 -> "Segundo cuadrante" // Caso: x < 0, y > 0
        x < 0 && y < 0 -> "Tercer cuadrante" // Caso: x < 0, y < 0
        x > 0 && y < 0 -> "Cuarto cuadrante" // Caso: x > 0, y < 0
        else -> "Un eje" // Caso: el punto está en un eje

    }

}

// Función principal que prueba los métodos de la clase Punto
fun main(parametro: Array<String>) {

    // Crea puntos en distintos cuadrantes y en un eje
    val punto1 = Punto(12, 3) // Punto en el primer cuadrante

    println("La coordenada ${punto1.x}, ${punto1.y} se encuentra en ${punto1.retornarCuadrante()}")

    val punto2 = Punto(-4, 3) // Punto en el segundo cuadrante

    println("La coordenada ${punto2.x}, ${punto2.y} se encuentra en ${punto2.retornarCuadrante()}")

    val punto3 = Punto(-2, -2) // Punto en el tercer cuadrante

    println("La coordenada ${punto3.x}, ${punto3.y} se encuentra en ${punto3.retornarCuadrante()}")

    val punto4 = Punto(12, -5) // Punto en el cuarto cuadrante

    println("La coordenada ${punto4.x}, ${punto4.y} se encuentra en ${punto4.retornarCuadrante()}")
}

```

```

    val punto5 = Punto(0, -5) // Punto en un eje

    println("La coordenada ${punto5.x}, ${punto5.y} se encuentra en
    ${punto5.retornarCuadrante()}")
}

```

**Documentación:** Este programa define una clase Punto que representa un punto en un plano cartesiano mediante las coordenadas x e y. Contiene un método que determina en qué cuadrante se encuentra el punto o si está sobre un eje. En la función principal main, se crean cinco objetos de la clase Punto para representar puntos en diferentes cuadrantes y en un eje, mostrando su ubicación.

### Proyecto 117:

// Clase Operaciones que define las propiedades y métodos para operaciones aritméticas

```

class Operaciones {

    var valor1: Int = 0 // Propiedad para almacenar el primer valor

    var valor2: Int = 0 // Propiedad para almacenar el segundo valor


    // Método para cargar los valores ingresados por el usuario
    fun cargar() {

        print("Ingrese primer valor:") // Solicita el primer valor

        valor1 = readLine()!!.toInt() // Lee y convierte el primer valor a entero


        print("Ingrese segundo valor:") // Solicita el segundo valor

        valor2 = readLine()!!.toInt() // Lee y convierte el segundo valor a entero


        sumar() // Llama al método sumar para realizar la operación

        restar() // Llama al método restar para realizar la operación

    }


    // Método para calcular y mostrar la suma de los valores
    fun sumar() {

        val suma = valor1 + valor2 // Realiza la suma de los dos valores
    }
}

```

```

        println("La suma de $valor1 y $valor2 es $suma") // Muestra el resultado de la suma
    }

    // Método para calcular y mostrar la resta de los valores
    fun restar() {
        val resta = valor1 - valor2 // Realiza la resta de los dos valores
        println("La resta de $valor1 y $valor2 es $resta") // Muestra el resultado de la resta
    }
}

// Función principal que ejecuta las operaciones definidas en la clase Operaciones
fun main(parametro: Array<String>) {
    val operaciones1 = Operaciones() // Crea un objeto de tipo Operaciones
    operaciones1.cargar() // Llama al método cargar para realizar la interacción y las operaciones
}

```

**Documentación:** Este programa define una clase llamada Operaciones, que permite realizar las operaciones básicas de suma y resta con dos números ingresados por el usuario. La clase contiene métodos para cargar los valores desde la entrada estándar, realizar la suma y la resta, e imprimir los resultados en la consola. En la función principal main, se crea un objeto de la clase Operaciones y se ejecuta su funcionalidad.

#### **Proyecto 118:**

```

// Clase Hijos que maneja las edades de cinco personas
class Hijos {
    val edades = IntArray(5) // Arreglo que almacena las edades de cinco personas

    // Método para cargar las edades desde la entrada estándar
    fun cargar() {
        for (i in edades.indices) { // Itera sobre cada índice del arreglo
            print("Ingresa edad:") // Solicita la edad por teclado
            edades[i] = readln().toInt() // Asigna la edad al índice correspondiente
        }
    }
}

```

```

    }

    mayorEdad() // Llama al método para calcular la mayor edad

    promedio() // Llama al método para calcular el promedio de edades
}

// Método para calcular y mostrar la mayor edad del arreglo
fun mayorEdad() {
    var mayor = edades[0] // Asume inicialmente que el primer valor es el mayor
    for (i in edades.indices) // Itera sobre el arreglo para encontrar el mayor
        if (edades[i] > mayor) // Compara cada valor con el actual mayor
            mayor = edades[i]
    println("El hijo con mayor edad es $mayor") // Muestra la mayor edad
}

// Método para calcular y mostrar el promedio de las edades
fun promedio() {
    var suma = 0 // Variable para acumular la suma de las edades
    for (i in edades.indices) // Itera sobre el arreglo para sumar las edades
        suma += edades[i]

    val promedio = suma / edades.size // Calcula el promedio dividiendo la suma entre la cantidad
    de elementos

    println("La edad promedio de los hijos es $promedio") // Muestra el promedio de las edades
}
}

// Función principal que prueba la funcionalidad de la clase Hijos
fun main(parametros: Array<String>) {
    val hijos1 = Hijos() // Crea un objeto de tipo Hijos
    hijos1.cargar() // Llama al método cargar para interactuar con el usuario y calcular los resultados
}

```

```
}
```

**Documentación:** Este programa define una clase llamada Hijos, que almacena las edades de cinco personas en un arreglo. La clase incluye métodos para cargar las edades desde la entrada estándar, determinar la mayor edad, y calcular el promedio de las edades. En la función principal main, se crea un objeto de la clase Hijos y se ejecutan sus métodos para interactuar con el usuario y mostrar los resultados.

#### **Proyecto 119:**

```
// Clase Cliente que representa a un cliente del banco
```

```
class Cliente(var nombre: String, var monto: Float) {
```

```
    // Método para depositar dinero en la cuenta del cliente
```

```
    fun depositar(monto: Float) {
```

```
        this.monto += monto // Incrementa el monto del cliente con el valor depositado
```

```
    }
```

```
    // Método para extraer dinero de la cuenta del cliente
```

```
    fun extraer(monto: Float) {
```

```
        this.monto -= monto // Decrementa el monto del cliente con el valor extraído
```

```
    }
```

```
    // Método para imprimir el saldo actual del cliente
```

```
    fun imprimir() {
```

```
        println("$nombre tiene depositado la suma de $monto") // Muestra el nombre y el saldo del cliente
```

```
    }
```

```
}
```

```
// Clase Banco que gestiona a múltiples clientes y sus operaciones
```

```
class Banco {
```

```
    val cliente1: Cliente = Cliente("Juan", 0f) // Cliente 1 con saldo inicial de 0
```

```

var cliente2: Cliente = Cliente("Ana", 0f) // Cliente 2 con saldo inicial de 0
var cliente3: Cliente = Cliente("Luis", 0f) // Cliente 3 con saldo inicial de 0

// Método que realiza operaciones de depósito y extracción
fun operar() {
    cliente1.depositar(100f) // Deposita 100 en el cliente 1
    cliente2.depositar(150f) // Deposita 150 en el cliente 2
    cliente3.depositar(200f) // Deposita 200 en el cliente 3
    cliente3.extraer(150f) // Extrae 150 del cliente 3
}

// Método que calcula y muestra el total acumulado en el banco y el estado de los clientes
fun depositosTotales() {
    val total = cliente1.monto + cliente2.monto + cliente3.monto // Suma los saldos de todos los
    clientes
    println("El total de dinero del banco es: $total") // Muestra el total acumulado
    cliente1.imprimir() // Muestra el estado del cliente 1
    cliente2.imprimir() // Muestra el estado del cliente 2
    cliente3.imprimir() // Muestra el estado del cliente 3
}
}

// Función principal que prueba la funcionalidad de las clases Cliente y Banco
fun main(parametro: Array<String>) {
    val banco1 = Banco() // Crea un objeto de tipo Banco
    banco1.operar() // Realiza las operaciones de depósito y extracción
    banco1.depositosTotales() // Calcula y muestra el total acumulado y el estado de los clientes
}

```



**Documentación:** Este programa define dos clases, Cliente y Banco, para simular un sistema básico de transacciones bancarias. La clase Cliente representa a un cliente del banco, mientras que Banco gestiona a varios clientes y realiza operaciones como depósitos, extracciones y cálculo del total acumulado en el banco. En la función principal main, se realizan estas operaciones y se muestra el estado de cada cliente.

**Proyecto 120:**

// Clase Dado que representa un dado con un valor asociado

```
class Dado(var valor: Int) {

    // Método para tirar el dado y asignar un valor aleatorio entre 1 y 6
    fun tirar() {
        valor = ((Math.random() * 6) + 1).toInt() // Genera un número aleatorio entre 1 y 6
        imprimir() // Llama al método imprimir para mostrar el valor
    }

    // Método para imprimir el valor actual del dado
    fun imprimir() {
        println("Valor del dado: $valor") // Muestra el valor del dado
    }
}
```

// Clase JuegoDeDados que gestiona el juego de los dados

```
class JuegoDeDados {
    val dado1 = Dado(1) // Instancia el primer dado con un valor inicial de 1
    val dado2 = Dado(1) // Instancia el segundo dado con un valor inicial de 1
    val dado3 = Dado(1) // Instancia el tercer dado con un valor inicial de 1

    // Método para jugar, tirando los tres dados y verificando si se gana o pierde
    fun jugar() {
        dado1.tirar() // Lanza el primer dado
    }
}
```

```

        dado2.tirar() // Lanza el segundo dado

        dado3.tirar() // Lanza el tercer dado


        // Verifica si los tres dados tienen el mismo valor
        if (dado1.valor == dado2.valor && dado2.valor == dado3.valor)
            println("Ganó") // Muestra "Ganó" si los valores son iguales
        else
            print("Perdió") // Muestra "Perdió" si no son iguales
    }
}

```

```

// Función principal que inicia el juego de dados
fun main(parametro: Array<String>) {
    val juego1 = JuegoDeDados() // Crea una instancia de JuegoDeDados
    juego1.jugar() // Llama al método jugar para realizar una partida
}

```

**Documentación:** Este programa consta de dos clases, Dado y JuegoDeDados, que simulan un juego donde se lanzan tres dados. Si los tres dados muestran el mismo valor, el jugador gana; de lo contrario, pierde. La clase Dado representa un dado individual, mientras que la clase JuegoDeDados gestiona los dados y la lógica del juego. En la función principal, se instancia el juego y se realiza una partida.

### Proyecto 121:

```

// Clase Socio que representa un miembro del club con nombre y antigüedad
class Socio(val nombre: String, val antigüedad: Int) {

    // Método para imprimir la información del socio
    fun imprimir() {
        println("$nombre tiene una antigüedad de $antigüedad") // Muestra el nombre y la antigüedad del socio
    }
}

```

```
// Clase Club que gestiona un conjunto de socios
```

```
class Club {
```

```
    val socio1 = Socio("Juan", 22) // Instancia el primer socio con nombre y antigüedad
```

```
    val socio2 = Socio("Ana", 34) // Instancia el segundo socio con nombre y antigüedad
```

```
    val socio3 = Socio("Carlos", 1) // Instancia el tercer socio con nombre y antigüedad
```

```
    // Método para determinar y mostrar el socio con mayor antigüedad
```

```
    fun mayorAntigüedad() {
```

```
        // Imprime la información de cada socio
```

```
        socio1.imprimir()
```

```
        socio2.imprimir()
```

```
        socio3.imprimir()
```

```
        // Determina y muestra el socio con mayor antigüedad
```

```
        print("Socio con mayor antigüedad:")
```

```
        when {
```

```
            socio1.antigüedad > socio2.antigüedad && socio1.antigüedad > socio3.antigüedad ->  
            print(socio1.nombre) // Caso: socio1 tiene mayor antigüedad
```

```
            socio2.antigüedad > socio3.antigüedad -> print(socio2.nombre) // Caso: socio2 tiene mayor  
            antigüedad
```

```
            else -> print(socio3.nombre) // Caso: socio3 tiene mayor antigüedad
```

```
        }
```

```
    }
```

```
}
```

```
// Función principal que prueba la funcionalidad de las clases Socio y Club
```

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

```
    val club1 = Club() // Crea una instancia de la clase Club
```

```

        club1.mayorAntigüedad() // Llama al método mayorAntigüedad para identificar al socio con mayor
        antigüedad
    }

```

**Documentación:** Este programa define dos clases, Socio y Club, para gestionar socios de un club y determinar quién tiene la mayor antigüedad. La clase Socio almacena información sobre cada socio, mientras que la clase Club contiene tres objetos de tipo Socio y un método para identificar al socio con mayor antigüedad en el club.

### Proyecto 122:

// Clase Operaciones2 que realiza operaciones aritméticas con dos valores

```

class Operaciones2 {

    // Propiedades privadas para almacenar los valores ingresados
    private var valor1: Int = 0
    private var valor2: Int = 0

    // Método público para cargar los valores desde el usuario
    fun cargar() {
        print("Ingrese primer valor:") // Solicita el primer valor
        valor1 = readLine()!!.toInt() // Lee y convierte el valor a entero

        print("Ingrese segundo valor:") // Solicita el segundo valor
        valor2 = readLine()!!.toInt() // Lee y convierte el valor a entero

        sumar() // Llama al método privado para calcular la suma
        restar() // Llama al método privado para calcular la resta
    }

    // Método privado para calcular y mostrar la suma de los valores
    private fun sumar() {
        val suma = valor1 + valor2 // Calcula la suma de los dos valores
        println("La suma de $valor1 y $valor2 es $suma") // Muestra el resultado de la suma
    }
}

```

```

    }

    // Método privado para calcular y mostrar la resta de los valores
    private fun restar() {
        val resta = valor1 - valor2 // Calcula la resta de los dos valores
        println("La resta de $valor1 y $valor2 es $resta") // Muestra el resultado de la resta
    }
}

// Función principal que prueba la funcionalidad de la clase Operaciones2
fun main(parametro: Array<String>) {
    val operaciones1 = Operaciones2() // Crea una instancia de la clase Operaciones2
    operaciones1.cargar() // Llama al método cargar para interactuar con el usuario y realizar las
operaciones
}

```

**Documentación:** Este programa define una clase llamada Operaciones2, que permite realizar operaciones básicas (suma y resta) de dos números ingresados por el usuario. Las propiedades y métodos relacionados con las operaciones son privados para garantizar que se accedan únicamente desde dentro de la clase. En la función principal main, se crea un objeto de la clase Operaciones2 y se ejecuta el método para cargar los valores y realizar las operaciones.

### Proyecto 123:

```

// Clase Dado2 que simula el funcionamiento de un dado
class Dado2 {
    private var valor: Int = 1 // Propiedad privada que almacena el valor actual del dado

    // Método para lanzar el dado y asignar un valor aleatorio entre 1 y 6
    fun tirar() {
        valor = ((Math.random() * 6) + 1).toInt() // Genera un número aleatorio entre 1 y 6
    }
}

```

```

// Método para imprimir el valor del dado con un separador decorativo
fun imprimir() {
    separador() // Llama al método separador antes de mostrar el valor
    println("Valor del dado: $valor") // Imprime el valor actual del dado
    separador() // Llama al método separador después de mostrar el valor
}

// Método privado que imprime un separador decorativo
private fun separador() = println("*****")
}

// Función principal que prueba la funcionalidad de la clase Dado2
fun main(parametro: Array<String>) {
    val dado1 = Dado2() // Crea una instancia de la clase Dado2
    dado1.tirar() // Llama al método tirar para lanzar el dado
    dado1.imprimir() // Llama al método imprimir para mostrar el resultado
}

```

**Documentación:** Este programa define una clase Dado2, que simula el lanzamiento de un dado y utiliza un formato visual con separadores para imprimir el resultado. La clase contiene métodos para tirar el dado, imprimir el valor y mostrar un separador decorativo. En la función principal main, se instancia el dado, se lanza y se imprime el resultado.

#### **Proyecto 124:**

```

// Clase Vector que gestiona un arreglo de enteros con operaciones adicionales
class Vector {
    private val vec = IntArray(5) // Propiedad privada: arreglo de 5 enteros

    // Bloque de inicialización que carga el arreglo con valores aleatorios
    init {
        cargar() // Llama al método cargar para llenar el arreglo al inicializar la clase
    }
}

```

```
// Método privado para cargar el arreglo con valores aleatorios entre 0 y 10
private fun cargar() {
    for (i in vec.indices) // Itera sobre los índices del arreglo
        vec[i] = ((Math.random() * 11)).toInt() // Genera un número aleatorio entre 0 y 10
}
```

```
// Método público para imprimir todos los elementos del arreglo
fun imprimir() {
    println("Listado completo del arreglo")
    for (i in vec.indices) // Itera sobre el arreglo para imprimir cada elemento
        print("${vec[i]} - ") // Imprime cada valor seguido de un guion
    println() // Salto de línea al final
}
```

```
// Método público para encontrar y mostrar el mayor elemento del arreglo
fun mostrarMayor() {
    var mayor = vec[0] // Asume inicialmente que el primer elemento es el mayor
    for (i in vec.indices) // Itera sobre el arreglo
        if (vec[i] > mayor) // Si encuentra un elemento mayor, lo actualiza
            mayor = vec[i]
    println("El elemento mayor del arreglo es $mayor") // Muestra el mayor valor
}
```

```
// Método público para encontrar y mostrar el menor elemento del arreglo
fun mostrarMenor() {
    var menor = vec[0] // Asume inicialmente que el primer elemento es el menor
    for (i in vec.indices) // Itera sobre el arreglo
        if (vec[i] < menor) // Si encuentra un elemento menor, lo actualiza
```

```

        menor = vec[i]

        println("El elemento menor del arreglo es $menor") // Muestra el menor valor
    }
}

```

```

// Función principal que prueba las funcionalidades de la clase Vector
fun main(parametro: Array<String>) {
    val vector1 = Vector() // Crea una instancia de la clase Vector
    vector1.imprimir() // Llama al método para imprimir el contenido del arreglo
    vector1.mostrarMayor() // Llama al método para mostrar el mayor valor
    vector1.mostrarMenor() // Llama al método para mostrar el menor valor
}

```

**Documentación:** Este programa crea una clase llamada Vector, que utiliza un arreglo de enteros para almacenar 5 valores aleatorios comprendidos entre 0 y 10. Los valores se generan automáticamente al inicializar la clase mediante un bloque init. La clase incluye métodos públicos para imprimir el arreglo completo, mostrar el mayor elemento y mostrar el menor elemento. En la función principal main, se crea una instancia de Vector y se ejecutan sus métodos.

### Proyecto 125:

```

import java.util.* // Importa utilidades para manipular texto, como uppercase

// Clase Persona3 que utiliza configuradores y accesorios personalizados
class Persona3 {
    var nombre: String = "" // Propiedad para almacenar el nombre

    // Configurador personalizado que convierte el nombre a mayúsculas
    set(valor) {
        field = valor.uppercase(Locale.getDefault()) // Convierte el valor a mayúsculas según la
        configuración regional
    }

    // Accesor personalizado que encapsula el nombre entre paréntesis
    get() {

```



```

        return "(\$field)" // Devuelve el nombre con paréntesis
    }

    var edad: Int = 0 // Propiedad para almacenar la edad

    // Configurador personalizado que valida que la edad no sea negativa
    set(valor) {
        if (valor >= 0) // Verifica que el valor sea mayor o igual a 0
            field = valor // Asigna el valor a la propiedad
        else
            field = 0 // Si el valor es negativo, asigna 0 como valor por defecto
    }
}

// Función principal que prueba las características de la clase Persona3
fun main(parametro: Array<String>) {
    val persona1 = Persona3() // Crea una instancia de Persona3

    persona1.nombre = "juan" // Asigna un nombre a la propiedad nombre
    persona1.edad = 23 // Asigna un valor válido a la propiedad edad

    println(persona1.nombre) // Imprime el nombre transformado a "(JUAN)"
    println(persona1.edad) // Imprime la edad: 23

    persona1.edad = -50 // Intenta asignar un valor negativo a la propiedad edad

    println(persona1.edad) // Imprime la edad, que se ajusta a 0 debido a la validación en el
    configurador
}

```

**Documentación:** Este programa define una clase `Persona3`, que utiliza propiedades con configuradores (`set`) y accesorios (`get`) personalizados para trabajar con validaciones y transformaciones de los datos. Las propiedades `nombre` y `edad` aplican lógica personalizada para

manejar la entrada y salida de valores. En la función principal main, se prueba el comportamiento de estas propiedades al asignar y mostrar diferentes valores.

#### **Proyecto 126:**

// Clase Empleado que representa a un empleado con nombre y sueldo

```
class Empleado(var nombre: String, sueldo: Double) {
```

```
    // Propiedad sueldo con configurador personalizado para evitar valores negativos
```

```
    var sueldo: Double = 0.0
```

```
    set(valor) {
```

```
        if (valor < 0) // Si el sueldo es negativo
```

```
            field = 0.0 // Asigna 0 como valor por defecto
```

```
        else
```

```
            field = valor // Asigna el valor si es válido
```

```
    }
```

```
    // Bloque de inicialización que asigna el sueldo al crear el objeto
```

```
    init {
```

```
        this.sueldo = sueldo // Asigna el sueldo inicial a través del configurador
```

```
    }
```

```
    // Método para imprimir el nombre y el sueldo del empleado
```

```
    fun imprimir() {
```

```
        println("$nombre tiene un sueldo de $sueldo") // Muestra el nombre y sueldo del empleado
```

```
    }
```

```
}
```

```
// Función principal que prueba las características de la clase Empleado
```

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

```
    val empleado1 = Empleado("Juan", 12000.50) // Crea un empleado con un sueldo válido
```

```

    empleado1.imprimir() // Llama al método imprimir para mostrar los datos del empleado

    val empleado2 = Empleado("Ana", -1200.0) // Crea un empleado con un sueldo negativo
    empleado2.imprimir() // Llama al método imprimir para mostrar que el sueldo se ajustó a 0
}

```

**Documentación:** Este programa define una clase Empleado, que gestiona el nombre y el sueldo de un empleado con validaciones para garantizar que el sueldo nunca sea negativo. Si se intenta asignar un valor negativo al sueldo, la clase ajusta el sueldo a 0. La clase también incluye un método para imprimir los detalles del empleado. En la función principal main, se crean y prueban dos instancias de la clase para validar su funcionalidad.

### Proyecto 127:

// Clase Dado4 que gestiona los valores de un dado con restricciones específicas

```

class Dado4(valor: Int) {

    var valor: Int = 1 // Propiedad que representa el valor actual del dado

    set(valor) {
        // Configurador que valida que el valor esté entre 1 y 6; si no, asigna 1
        if (valor >= 1 && valor <= 6)
            field = valor // Asigna el valor si es válido
        else
            field = 1 // Asigna 1 como valor por defecto si es inválido
    }

    // Bloque init que inicializa el dado con el valor recibido en el constructor
    init {
        this.valor = valor // Usa el configurador para validar y asignar el valor inicial
    }

    // Método para asignar un valor aleatorio entre 1 y 6 al dado
    fun tirar() {
        valor = ((Math.random() * 6) + 1).toInt() // Genera un número aleatorio entre 1 y 6
    }
}

```

```

    }

    // Método para imprimir el valor actual del dado
    fun imprimir() = println("Valor del dado: $valor")
}

// Función principal que prueba las funcionalidades de la clase Dado4
fun main(parametro: Array<String>) {
    val dado1 = Dado4(7) // Crea un objeto de la clase Dado4 con un valor inicial fuera del rango permitido
    dado1.imprimir() // Imprime el valor inicial del dado, ajustado por el configurador

    dado1.tirar() // Lanza el dado para generar un valor aleatorio
    dado1.imprimir() // Imprime el valor aleatorio generado
}

```

**Documentación:** Este programa define una clase llamada Dado4 que gestiona un dado con reglas específicas:

- La propiedad valor asegura que el valor asignado esté en el rango de 1 a 6. Si el valor está fuera de este rango, se asigna automáticamente un valor de 1.
- La clase incluye un método para generar un valor aleatorio entre 1 y 6 (tirar) y otro para imprimir el valor actual del dado (imprimir).
- El constructor inicial permite asignar un valor al dado, pero aplica las restricciones definidas por el configurador.

En la función principal main, se prueba el comportamiento del dado al recibir un valor inicial fuera del rango permitido y al generar un valor aleatorio.

### Proyecto 128:

```

// Definición de una data class Artículo con propiedades mutables
data class Artículo(var codigo: Int, var descripcion: String, var precio: Float)

fun main(parametro: Array<String>) {
    // Crea un objeto de tipo Artículo con valores iniciales
}

```

```
val articulo1 = Artículo(1, "papas", 34f)
var articulo2 = Artículo(2, "manzanas", 24f)

// Imprime los datos de los artículos
println(articulo1) // Muestra: Artículo(codigo=1, descripcion=papas, precio=34.0)
println(articulo2) // Muestra: Artículo(codigo=2, descripcion=manzanas, precio=24.0)

// Crea una referencia al objeto articulo1
val puntero = articulo1

// Modifica el precio del objeto apuntado por 'puntero', que afecta también a articulo1
puntero.precio = 100f
println(articulo1) // Muestra: Artículo(codigo=1, descripcion=papas, precio=100.0)

// Crea una copia de articulo1
var articulo3 = articulo1.copy()

// Modifica el precio de articulo1, pero no afecta a articulo3 (ya que es una copia)
articulo1.precio = 200f
println(articulo1) // Muestra: Artículo(codigo=1, descripcion=papas, precio=200.0)
println(articulo3) // Muestra: Artículo(codigo=1, descripcion=papas, precio=100.0)

// Compara si articulo1 y articulo3 son iguales después del cambio
if (articulo1 == articulo3) // Compara las propiedades (no las referencias)
    println("Son iguales $articulo1 y $articulo3")
else
    println("Son distintos $articulo1 y $articulo3") // Muestra: Son distintos

// Modifica el precio de articulo3 para que coincida con articulo1
articulo3.precio = 200f

// Compara nuevamente los dos objetos
```

```

if (articulo1 == articulo3)

    println("Son iguales $articulo1 y $articulo3") // Muestra: Son iguales

else

    println("Son distintos $articulo1 y $articulo3")

}

```

**Documentación:** Este programa utiliza una data class llamada Articulo para representar artículos con las propiedades código, descripción y precio. En la función principal, se realizan varias operaciones para manipular objetos de esta clase, incluyendo copias, modificaciones y comparaciones entre objetos.

#### Proyecto 129:

```

// Data class Persona4 con propiedades nombre y edad

data class Persona4(var nombre: String, var edad: Int) {

    // Sobrescribe el método toString para personalizar la salida como "nombre, edad"

    override fun toString(): String {

        return "$nombre, $edad" // Retorna una representación personalizada del objeto

    }

}

fun main(parametro: Array<String>) {

    // Crea una instancia de Persona4 para representar a Juan

    var persona1 = Persona4("Juan", 22)

    // Crea una instancia de Persona4 para representar a Ana

    var persona2 = Persona4("Ana", 59)

    // Imprime la representación en cadena de persona1 y persona2

    println(persona1) // Salida esperada: "Juan, 22"

    println(persona2) // Salida esperada: "Ana, 59"

}

```

**Documentación:** Este programa utiliza una data class llamada Persona4 que almacena las propiedades nombre y edad. Se sobrescribe el método toString para personalizar la representación en

forma de cadena de los objetos de esta clase. En la función main, se crean e imprimen dos instancias de la clase Persona4.

#### **Proyecto 130:**

// Data class Dados que representa un dado con su valor

```
data class Dados(var valor: Int) {  
    // Sobrescribe el método toString para generar una cadena de asteriscos basada en el valor  
    override fun toString(): String {  
        var cadena = "" // Variable para construir la cadena de asteriscos  
        for (i in 1..valor) // Itera desde 1 hasta el valor del dado  
            cadena = cadena + "*" // Agrega un asterisco por cada iteración  
        return cadena // Retorna la cadena de asteriscos  
    }  
}
```

// Función principal que prueba la funcionalidad del data class Dados

```
fun main(parametro: Array<String>) {  
    val dado1 = Dados(4) // Instancia un dado con valor 4  
    val dado2 = Dados(6) // Instancia un dado con valor 6  
    val dado3 = Dados(1) // Instancia un dado con valor 1  
  
    // Imprime las representaciones personalizadas de los objetos Dados  
    println(dado1) // Muestra: ****  
    println(dado2) // Muestra: *****  
    println(dado3) // Muestra: *  
}
```

**Documentación:** Este programa define un data class llamado Dados, que representa un dado con una única propiedad valor. El método toString se sobrescribe para generar una cadena que contiene tantos asteriscos como el valor del dado. En la función principal, se crean tres instancias del dado con diferentes valores y se imprimen los resultados utilizando la lógica personalizada de toString.

#### **Proyecto 131:**

```
// Enumeración que define los tipos de cartas en una baraja
```

```
enum class TipoCarta {  
    DIAMANTE, // Representa el tipo "Diamante"  
    TREBOL,   // Representa el tipo "Trébol"  
    CORAZON,  // Representa el tipo "Corazón"  
    PICA      // Representa el tipo "Pica"  
}
```

```
// Clase que representa una carta con su tipo y valor
```

```
class Carta(val tipo: TipoCarta, val valor: Int) {  
    // Método para imprimir los detalles de la carta  
    fun imprimir() {  
        println("Carta: $tipo y su valor es $valor") // Muestra el tipo y el valor de la carta  
    }  
}
```

```
// Función principal que prueba la funcionalidad de la clase Carta y la enumeración TipoCarta
```

```
fun main(parametro: Array<String>) {  
    val carta1 = Carta(TipoCarta.TREBOL, 4) // Crea una carta de tipo "Trébol" con valor 4  
    carta1.imprimir() // Llama al método imprimir para mostrar los detalles de la carta  
}
```

**Documentación:** Este programa utiliza una enumeración TipoCarta y una clase Carta para representar cartas de una baraja con su tipo y valor. La enumeración define cuatro posibles tipos de cartas: DIAMANTE, TREBOL, CORAZON y PICA. La clase Carta contiene las propiedades necesarias para identificar el tipo y valor de una carta, además de un método para imprimir la descripción de la carta. En la función main, se crea una instancia de la clase Carta y se utiliza el método para mostrar sus detalles.

### Proyecto 132:

```
// Enumeración que define los tipos de operaciones matemáticas
```

```
enum class TipoOperacionn(val tipo: String) {
```



```

SUMA("+"),      // Representa la operación de suma
RESTA("-"),     // Representa la operación de resta
MULTIPLICACION("*"), // Representa la operación de multiplicación
DIVISION("/")   // Representa la operación de división
}

// Clase Operacion que representa una operación matemática entre dos valores
class Operacion(val valor1: Int, val valor2: Int, val tipoOperacion: TipoOperacionn) {
    // Método para realizar la operación y mostrar el resultado
    fun operar() {
        var resultado: Int = 0 // Variable para almacenar el resultado de la operación
        when (tipoOperacion) { // Evalúa el tipo de operación
            TipoOperacionn.SUMA -> resultado = valor1 + valor2      // Realiza la suma
            TipoOperacionn.RESTA -> resultado = valor1 - valor2     // Realiza la resta
            TipoOperacionn.MULTIPLICACION -> resultado = valor1 * valor2 // Realiza la multiplicación
            TipoOperacionn.DIVISION -> resultado = valor1 / valor2   // Realiza la división
        }
        // Muestra la operación realizada y el resultado
        println("$valor1 ${tipoOperacion.tipo} $valor2 es igual a $resultado")
    }
}

// Función principal que prueba las operaciones matemáticas
fun main(parametro: Array<String>) {
    // Instancia una operación de suma
    val operacion1 = Operacion(10, 4, TipoOperacionn.SUMA)

    // Instancia una operación de resta
    val operacion2 = Operacion(10, 4, TipoOperacionn.RESTA)

```

```

// Instancia una operación de multiplicación
val operacion3 = Operacion(10, 4, TipoOperacionn.MULTIPLICACION)

// Instancia una operación de división
val operacion4 = Operacion(10, 4, TipoOperacionn.DIVISION)


// Realiza las operaciones y muestra los resultados
operacion1.operar() // Muestra: "10 + 4 es igual a 14"
operacion2.operar() // Muestra: "10 - 4 es igual a 6"
operacion3.operar() // Muestra: "10 * 4 es igual a 40"
operacion4.operar() // Muestra: "10 / 4 es igual a 2"
}

```

**Documentación:** Este programa utiliza una enumeración TipoOperacionn para definir los tipos de operaciones matemáticas básicas: suma, resta, multiplicación y división. La clase Operacion representa una operación matemática específica con dos valores y un tipo de operación. Incluye un método operar que realiza la operación según el tipo especificado. En la función principal main, se crean varias instancias de la clase Operacion para realizar y mostrar diferentes cálculos.

### Proyecto 133:

```

// Enumeración que define países sudamericanos y sus habitantes
enum class Países(val habitantes: Int) {

    BRASIL(202450649),    // Brasil con 202,450,649 habitantes
    COLOMBIA(50364000),   // Colombia con 50,364,000 habitantes
    PERU(31151643),       // Perú con 31,151,643 habitantes
    VENEZUELA(31028337),  // Venezuela con 31,028,337 habitantes
    CHILE(18261884),      // Chile con 18,261,884 habitantes
    ECUADOR(16298217),    // Ecuador con 16,298,217 habitantes
    BOLIVIA(10888000),    // Bolivia con 10,888,000 habitantes
    PARAGUAY(6460000),    // Paraguay con 6,460,000 habitantes
    URUGUAY(3372000)      // Uruguay con 3,372,000 habitantes
}

```

// Función principal que imprime los países y sus habitantes

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

```
    // Ejemplo con Brasil
```

```
    val pais1 = Paises.BRASIL
```

```
    println(pais1)      // Imprime: BRASIL
```

```
    println(pais1.habitantes) // Imprime: 202450649
```

```
    // Ejemplo con Colombia
```

```
    val pais2 = Paises.COLOMBIA
```

```
    println(pais2)      // Imprime: COLOMBIA
```

```
    println(pais2.habitantes) // Imprime: 50364000
```

```
    // Ejemplo con Perú
```

```
    val pais3 = Paises.PERU
```

```
    println(pais3)      // Imprime: PERU
```

```
    println(pais3.habitantes) // Imprime: 31151643
```

```
    // Ejemplo con Venezuela
```

```
    val pais4 = Paises.VENEZUELA
```

```
    println(pais4)      // Imprime: VENEZUELA
```

```
    println(pais4.habitantes) // Imprime: 31028337
```

```
    // Ejemplo con Chile
```

```
    val pais5 = Paises.CHILE
```

```
    println(pais5)      // Imprime: CHILE
```

```
    println(pais5.habitantes) // Imprime: 18261884
```

```
    // Ejemplo con Ecuador
```

```
    val pais6 = Paises.ECUADOR
```

```

println(pais6)      // Imprime: ECUADOR
println(pais6.habitantes) // Imprime: 16298217

// Ejemplo con Bolivia
val pais7 = Paises.BOLIVIA
println(pais7)      // Imprime: BOLIVIA
println(pais7.habitantes) // Imprime: 10888000

// Ejemplo con Paraguay
val pais8 = Paises.PARAGUAY
println(pais8)      // Imprime: PARAGUAY
println(pais8.habitantes) // Imprime: 6460000

// Ejemplo con Uruguay
val pais9 = Paises.URUGUAY
println(pais9)      // Imprime: URUGUAY
println(pais9.habitantes) // Imprime: 3372000
}

```

**Documentación:** Este programa utiliza un enum class llamado Paises, donde cada constante representa un país sudamericano y tiene asociada una propiedad habitantes que almacena la cantidad de habitantes de ese país. En la función main, se crean variables para cada país, se imprimen las constantes y sus respectivas cantidades de habitantes.

#### **Proyecto 134:**

```

// Objeto Matematica que contiene propiedades y métodos relacionados con matemáticas
object Matematica {

    val PI = 3.1416 // Propiedad que almacena el valor aproximado de Pi

    // Función que genera un número aleatorio entre 'minimo' y 'maximo' (ambos inclusivos)

    fun aleatorio(minimo: Int, maximo: Int) = ((Math.random() * (maximo + 1 - minimo)) +
minimo).toInt()

```

```
}
```

```
// Función principal para probar el objeto Matematica
```

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

```
    // Imprime el valor de Pi desde el objeto Matematica
```

```
    println("El valor de Pi es ${Matematica.PI}")
```

```
    // Genera un número aleatorio entre 5 y 10 y lo imprime
```

```
    print("Un valor aleatorio entre 5 y 10: ")
```

```
    println(Matematica.aleatorio(5, 10))
```

```
}
```

**Documentación:** Este programa utiliza el objeto Matematica para encapsular el valor de  $\pi$  (PI) y una función para generar números aleatorios en un rango especificado. En la función principal, se utiliza este objeto para mostrar el valor de  $\pi$  y generar un número aleatorio entre 5 y 10.

**Proyecto 135:**

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

```
    // Definición de un objeto anónimo que representa un conjunto de dados
```

```
    val dados = object {
```

```
        val arreglo = IntArray(5) // Arreglo para almacenar los valores de los dados
```

```
        // Método para generar valores aleatorios entre 1 y 6 y asignarlos al arreglo
```

```
        fun generar() {
```

```
            for (i in arreglo.indices) // Itera sobre los índices del arreglo
```

```
                arreglo[i] = ((Math.random() * 6) + 1).toInt() // Genera un número aleatorio entre 1 y 6
```

```
        }
```

```
        // Método para imprimir los valores del arreglo
```

```
        fun imprimir() {
```

```
            for (elemento in arreglo) // Itera sobre los valores del arreglo
```

```

        print("$elemento - ") // Imprime cada valor separado por guiones
        println() // Salto de línea al final
    }

    // Método para encontrar el mayor valor en el arreglo
    fun mayor(): Int {
        var may = arreglo[0] // Asume inicialmente que el primer elemento es el mayor
        for (i in arreglo.indices) // Itera sobre el arreglo
            if (arreglo[i] > may) // Si encuentra un valor mayor, lo actualiza
                may = arreglo[i]
        return may // Retorna el mayor valor encontrado
    }
}

// Llama al método para generar valores aleatorios
dados.generar()

// Imprime el contenido del arreglo
dados.imprimir()

// Calcula e imprime el mayor valor
print("Mayor valor:")
println(dados.mayor())
}

```

**Documentación:** Este programa utiliza un objeto anónimo para gestionar cinco números aleatorios que simulan lanzamientos de un dado. El objeto contiene un arreglo para almacenar los valores, un método para generarlos aleatoriamente, otro para imprimirlos, y un tercero para determinar el mayor valor. La funcionalidad del objeto se ejecuta en el método main.

#### **Proyecto 136:**

```

fun main() {
    println("Ingrese el primer número:")

    val num1 = readLine()!!.toInt() // Solicita el primer número
}

```

```
println("Ingrese el segundo número:")

val num2 = readLine()!!.toInt() // Solicita el segundo número

val suma = num1 + num2 // Realiza la suma

println("La suma es: $suma") // Presenta el resultado

}
```

**Documentación:** Este programa realiza la suma de dos números proporcionados por el usuario. Incluye comentarios que describen la lógica paso a paso. La función principal solicita los números, realiza la operación, y presenta el resultado en la consola.

#### Proyecto 137:

```
fun main() {

    println("Ingrese un año:")

    val year = readLine()!!.toInt() // Solicita el año

    if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)) {

        println("$year es un año bisiesto") // Presenta resultado positivo

    } else {

        println("$year no es un año bisiesto") // Presenta resultado negativo

    }

}
```

**Documentación:** Este programa determina si un año ingresado por el usuario es bisiesto. La lógica utiliza operadores condicionales para evaluar si cumple con los criterios de divisibilidad. El resultado se imprime en la consola para mayor claridad.

#### Proyecto 138:

```
fun main() {

    println("Ingrese el límite inferior:")

    val min = readLine()!!.toInt() // Solicita límite inferior

    println("Ingrese el límite superior:")

    val max = readLine()!!.toInt() // Solicita límite superior

    val aleatorio = ((Math.random() * (max - min + 1)) + min).toInt() // Calcula aleatorio

    println("Número aleatorio generado: $aleatorio") // Presenta el número

}
```

**Documentación:** Este programa genera un número aleatorio entre dos valores definidos por el usuario. Utiliza la función `Math.random()` para la lógica de aleatoriedad y asegura que los límites sean inclusivos en el cálculo.

**Proyecto 139:**

```
fun main() {  
  
    println("Ingrese un número para calcular su factorial:")  
  
    val numero = readLine()!!.toInt() // Solicita el número  
  
    var factorial = 1  
  
    for (i in 1..numero) factorial *= i // Calcula el factorial  
  
    println("El factorial de $numero es: $factorial") // Muestra el resultado  
  
}
```

**Documentación:** Este programa calcula el factorial de un número ingresado por el usuario. Utiliza un ciclo `for` para multiplicar todos los números desde 1 hasta el número ingresado, mostrando el resultado en la consola.

**Proyecto 140:**

```
fun main() {  
  
    println("Ingrese un número para clasificarlo:")  
  
    val numero = readLine()!!.toInt() // Solicita el número  
  
    when {  
  
        numero > 0 -> println("$numero es positivo") // Caso positivo  
        numero < 0 -> println("$numero es negativo") // Caso negativo  
        else -> println("El número es cero") // Caso cero  
  
    }  
  
}
```

**Documentación:** Este programa evalúa si un número ingresado por el usuario es positivo, negativo o cero. Utiliza una estructura condicional para determinar la clasificación y presenta el resultado en la consola.

**Proyecto 141:**

```
fun main() {  
  
    println("Ingrese un número para generar su tabla de multiplicar:")  
  
    val numero = readLine()!!.toInt() // Solicita el número
```



```

    for (i in 1..10) println("$numero x $i = ${numero * i}") // Calcula y muestra la tabla
}

```

**Documentación:** Este programa genera la tabla de multiplicar de un número ingresado por el usuario. Utiliza un ciclo for para calcular los productos hasta el 10 y muestra cada resultado en la consola.

#### Proyecto 142:

```

fun main() {
    println("Ingresa la temperatura en Celsius:")
    val celsius = readLine()!!.toDouble() // Solicita la temperatura en Celsius
    val fahrenheit = (celsius * 9 / 5) + 32 // Calcula Fahrenheit
    println("$celsius°C son equivalentes a $fahrenheit°F") // Muestra el resultado
}

```

**Documentación:** Este programa convierte una temperatura de Celsius a Fahrenheit utilizando la fórmula correspondiente. La temperatura en Celsius es ingresada por el usuario y el resultado se muestra en la consola.

#### Proyecto 143:

```

fun main() {
    println("Ingresa la cantidad de números:")
    val cantidad = readLine()!!.toInt() // Solicita la cantidad de números
    val numeros = IntArray(cantidad) // Crea el arreglo
    for (i in 0 until cantidad) {
        println("Ingresa el número ${i + 1}:")
        numeros[i] = readLine()!!.toInt() // Llena el arreglo
    }
    val promedio = numeros.average() // Calcula el promedio
    println("El promedio es: $promedio") // Muestra el resultado
}

```

**Documentación:** Este programa calcula el promedio de un conjunto de números ingresados por el usuario. Utiliza un arreglo para almacenar los números y una función de suma para calcular el promedio.

#### Proyecto 144:

```

fun main() {

    println("Ingrese el radio del círculo:")

    val radio = readLine()!!.toDouble() // Solicita el radio

    val area = 3.1416 * radio * radio // Calcula el área usando la fórmula

    println("El área del círculo es: $area") // Muestra el resultado

}

```

**Documentación:** Este programa calcula el área de un círculo con base en el radio proporcionado por el usuario. Utiliza la fórmula matemática  $PI * radio^2$  y muestra el resultado en la consola, asegurando claridad en la representación.

#### Proyecto 145:

```

fun main() {

    println("Ingrese la distancia en kilómetros:")

    val km = readLine()!!.toDouble() // Solicita la distancia en kilómetros

    val millas = km * 0.621371 // Convierte a millas

    println("$km kilómetros son equivalentes a $millas millas") // Muestra el resultado

}

```

**Documentación:** Este programa convierte una distancia en kilómetros a millas utilizando la fórmula  $kilómetros * 0.621371$ . El usuario proporciona la distancia en kilómetros y el resultado es impreso en la consola.

#### Proyecto 146:

```

fun main() {

    println("Ingrese un número:")

    val numero = readLine()!!.toInt() // Solicita el número

    if (numero % 2 == 0) {

        println("$numero es par") // Resultado para números pares

    } else {

        println("$numero es impar") // Resultado para números impares

    }

}

```

**Documentación:** Este programa determina si un número ingresado por el usuario es par o impar. Utiliza el operador módulo para verificar la divisibilidad entre 2 y muestra el resultado en la consola.

**Proyecto 147:**

```
fun main() {  
    println("Ingresa el primer cateto:")  
    val cateto1 = readLine()!!.toDouble() // Solicita el primer cateto  
    println("Ingresa el segundo cateto:")  
    val cateto2 = readLine()!!.toDouble() // Solicita el segundo cateto  
    val hipotenusa = kotlin.math.sqrt(cateto1 * cateto1 + cateto2 * cateto2) // Calcula la hipotenusa  
    println("La hipotenusa es: $hipotenusa") // Muestra el resultado  
}
```

**Documentación:** Este programa calcula la hipotenusa de un triángulo rectángulo utilizando el teorema de Pitágoras. El usuario ingresa las longitudes de los catetos y el programa imprime el resultado calculado.

**Proyecto 148:**

```
fun main() {  
    println("Ingresa su edad:")  
    val edad = readLine()!!.toInt() // Solicita la edad  
    if (edad >= 18) {  
        println("Eres apto para votar") // Caso positivo  
    } else {  
        println("No eres apto para votar") // Caso negativo  
    }  
}
```

**Documentación:** Este programa verifica si una persona es apta para votar según su edad. Solicita la edad del usuario y utiliza una estructura condicional para determinar si cumple con el requisito legal.

**Proyecto 149:**

```
fun main() {  
    println("Ingresa el radio del cilindro:")  
    val radio = readLine()!!.toDouble() // Solicita el radio
```

```
println("Ingrese la altura del cilindro:")

val altura = readLine()!!.toDouble() // Solicita la altura

val volumen = 3.1416 * radio * radio * altura // Calcula el volumen

println("El volumen del cilindro es: $volumen") // Muestra el resultado

}
```

**Documentación:** Este programa calcula el volumen de un cilindro utilizando la fórmula  $PI * radio^2 * altura$ . Solicita el radio y la altura como entradas, realiza el cálculo, y muestra el resultado en la consola de manera clara.

#### Proyecto 150:

```
fun main() {

    println("Ingrese el número de horas:")

    val horas = readLine()!!.toInt() // Solicita las horas

    val segundos = horas * 3600 // Convierte a segundos

    println("$horas horas son equivalentes a $segundos segundos") // Presenta el resultado

}
```

**Documentación:** Este programa convierte una cantidad de horas ingresada por el usuario a segundos. Utiliza la fórmula  $horas * 3600$  para realizar la conversión y muestra el resultado en la consola.

#### Proyecto 151:

```
fun main() {

    val contrasenaCorrecta = "abc123" // Define la contraseña correcta

    println("Ingrese la contraseña:")

    val contrasena = readLine()!! // Solicita la contraseña

    if (contrasena == contrasenaCorrecta) {

        println("Contraseña correcta") // Caso exitoso

    } else {

        println("Contraseña incorrecta") // Caso fallido

    }

}
```

**Documentación:** Este programa valida si una contraseña ingresada por el usuario cumple con un criterio predefinido. Utiliza una estructura condicional para comparar la entrada con la contraseña correcta y muestra un mensaje adecuado.

**Proyecto 152:**

```
fun main() {  
    println("Ingresa un año:")  
    val year = readLine()!!.toInt() // Solicita el año  
    if (year % 100 == 0) {  
        println("$year es divisible por 100") // Presenta resultado positivo  
    } else {  
        println("$year no es divisible por 100") // Presenta resultado negativo  
    }  
}
```

**Documentación:** Este programa determina si un año ingresado por el usuario es divisible por 100. Utiliza el operador módulo para comprobar la divisibilidad y presenta el resultado en la consola de forma clara.

**Proyecto 153:**

```
fun main() {  
    println("¿Cuántos números Fibonacci desea generar?")  
    val n = readLine()!!.toInt() // Solicita el número de elementos  
    var a = 0  
    var b = 1  
    print("$a $b ") // Imprime los dos primeros números  
    for (i in 3..n) {  
        val c = a + b  
        print("$c ") // Muestra el siguiente número  
        a = b  
        b = c  
    }  
    println() // Salto de línea al final
```

```
}
```

**Documentación:** Este programa genera los primeros n números de la secuencia Fibonacci, donde n es proporcionado por el usuario. Utiliza un ciclo para calcular cada número de la secuencia y muestra los resultados en la consola.

**Proyecto 154:**

```
fun main() {  
    println("Ingrese la base del triángulo:")  
    val base = readLine()!!.toDouble() // Solicita la base  
    println("Ingrese la altura del triángulo:")  
    val altura = readLine()!!.toDouble() // Solicita la altura  
    val area = (base * altura) / 2 // Calcula el área  
    println("El área del triángulo es: $area") // Muestra el resultado  
}
```

**Documentación:** Este programa calcula el área de un triángulo basándose en la fórmula  $(base * altura) / 2$ . Solicita la base y la altura como entradas del usuario, realiza el cálculo y muestra el resultado en la consola de forma precisa.

**Proyecto 155:**

```
fun main() {  
    println("Ingrese una cadena de texto:")  
    val texto = readLine()!! // Solicita el texto  
    val invertido = texto.reversed() // Invierte la cadena  
    println("Texto invertido: $invertido") // Presenta el resultado  
}
```

**Documentación:** Este programa invierte una cadena de texto ingresada por el usuario. Utiliza la función `reversed()` para transformar la cadena y muestra la inversión en la consola.

**Proyecto 156:**

```
fun main() {  
    println("Ingrese una palabra:")  
    val palabra = readLine()!! // Solicita la palabra  
    val esPalindromo = palabra == palabra.reversed() // Verifica si es un palíndromo
```

```

    if (esPalindromo) println("$palabra es un palíndromo") // Caso positivo
    else println("$palabra no es un palíndromo") // Caso negativo
}

```

**Documentación:** Este programa verifica si una cadena de texto ingresada por el usuario es un palíndromo. Compara la cadena original con su versión invertida y muestra el resultado en la consola.

#### **Proyecto 157:**

```

fun main() {
    println("Ingrese la cantidad de elementos:")
    val cantidad = readLine()!!.toInt() // Solicita la cantidad de elementos
    val arreglo = IntArray(cantidad) // Crea el arreglo
    for (i in 0 until cantidad) {
        println("Ingrese el elemento ${i + 1}:")
        arreglo[i] = readLine()!!.toInt() // Llena el arreglo
    }
    val suma = arreglo.sum() // Calcula la suma de los elementos
    println("La suma de los elementos es: $suma") // Muestra el resultado
}

```

**Documentación:** Este programa calcula la suma de los elementos de un arreglo. El usuario ingresa los valores que conforman el arreglo, y el programa utiliza la función `sum()` para calcular la suma total.

#### **Proyecto 158:**

```

fun main() {
    println("Ingrese el número de personas:")
    val personas = readLine()!!.toInt() // Solicita el número de personas
    val edades = IntArray(personas) // Crea el arreglo para edades
    for (i in 0 until personas) {
        println("Ingrese la edad de la persona ${i + 1}:")
        edades[i] = readLine()!!.toInt() // Llena el arreglo con edades
    }
    val media = edades.average() // Calcula la media de las edades
}

```

```
println("La media de las edades es: $media") // Muestra el resultado
}
```

**Documentación:** Este programa calcula la media de las edades ingresadas por un conjunto de personas. Solicita las edades como entrada, las almacena en un arreglo, calcula la media, y presenta el resultado en la consola.

#### Proyecto 159:

```
fun main() {
    println("Ingrese la cantidad de números:")
    val cantidad = readLine()!!.toInt() // Solicita la cantidad de números
    val numeros = IntArray(cantidad) // Crea el arreglo
    for (i in 0 until cantidad) {
        println("Ingrese el número ${i + 1}:")
        numeros[i] = readLine()!!.toInt() // Llena el arreglo
    }
    val minimo = numeros.minOrNull() // Encuentra el mínimo
    val maximo = numeros.maxOrNull() // Encuentra el máximo
    println("El número mínimo es: $minimo y el máximo es: $maximo") // Muestra los resultados
}
```

**Documentación:** Este programa identifica el número mínimo y el máximo dentro de un arreglo de enteros ingresados por el usuario. Usa ciclos para recorrer el arreglo y compara cada elemento para encontrar los valores deseados, mostrando el resultado en la consola.

#### Proyecto 160:

```
fun main() {
    println("Ingrese la cantidad de notas:")
    val cantidad = readLine()!!.toInt() // Solicita la cantidad de notas
    val notas = DoubleArray(cantidad) // Crea el arreglo para las notas
    for (i in 0 until cantidad) {
        println("Ingrese la nota ${i + 1}:")
        notas[i] = readLine()!!.toDouble() // Llena el arreglo
    }
}
```



```

    val suma = notas.sum() // Calcula la suma de las notas

    val promedio = suma / cantidad // Calcula el promedio

    println("La suma de las notas es: $suma y el promedio es: $promedio") // Muestra los resultados
}

```

**Documentación:** Este programa calcula la suma y el promedio de un conjunto de calificaciones ingresadas por el usuario. Usa un arreglo para almacenar las notas, calcula la suma y el promedio, y muestra los resultados en la consola.

#### Proyecto 161:

```

fun main() {

    println("Ingrese una cadena de texto:")

    val texto = readLine()!! // Solicita el texto

    val cantidad = texto.length // Calcula la cantidad de caracteres

    println("La cantidad de caracteres es: $cantidad") // Presenta el resultado
}

```

**Documentación:** Este programa cuenta cuántos caracteres tiene una cadena de texto ingresada por el usuario. Utiliza la propiedad length de las cadenas en Kotlin para realizar el cálculo y presenta el resultado en la consola.

#### Proyecto 162:

```

fun main() {

    println("Ingrese un número:")

    val numero = readLine()!!.toInt() // Solicita el número

    var esPrimo = true

    if (numero < 2) esPrimo = false // Caso especial para números menores a 2

    for (i in 2 until numero) {

        if (numero % i == 0) {

            esPrimo = false // Encuentra un divisor

            break

        }

    }

    if (esPrimo) println("$numero es un número primo") // Caso positivo
}

```

```
        else println("$numero no es un número primo") // Caso negativo
    }
}
```

**Documentación:** Este programa verifica si un número ingresado por el usuario es primo. Usa un ciclo para comprobar si el número tiene divisores, determinando así si es primo o no, y muestra el resultado en la consola.

**Proyecto 163:**

```
fun main() {
    println("Ingrese la base:")
    val base = readLine()!!.toDouble() // Solicita la base
    println("Ingrese el exponente:")
    val exponente = readLine()!!.toDouble() // Solicita el exponente
    val resultado = Math.pow(base, exponente) // Calcula la potencia
    println("$base elevado a la $exponente es igual a $resultado") // Presenta el resultado
}
```

**Documentación:** Este programa calcula la potencia de un número base elevado a un exponente, ambos ingresados por el usuario. Utiliza la función `Math.pow` y muestra el resultado en la consola.

**Proyecto 164:**

```
fun main() {
    println("Ingrese un número para calcular su raíz cuadrada:")
    val numero = readLine()!!.toDouble() // Solicita el número
    val raiz = Math.sqrt(numero) // Calcula la raíz cuadrada
    println("La raíz cuadrada de $numero es: $raiz") // Muestra el resultado
}
```

**Documentación:** Este programa calcula la raíz cuadrada de un número proporcionado por el usuario. Utiliza la función `Math.sqrt` para realizar el cálculo y muestra el resultado en la consola de manera clara y precisa.

**Proyecto 165:**

```
fun main() {
    println("Ingrese una palabra:")
    val palabra = readLine()!! // Solicita la palabra
}
```

```

println("¿Cuántas veces desea repetirla?")

val repeticiones = readLine()!!.toInt() // Solicita el número de repeticiones

for (i in 1..repeticiones) {
    println("$i. $palabra") // Muestra la palabra repetida
}
}

```

**Documentación:** Este programa toma una palabra ingresada por el usuario y la repite un número de veces también definido por el usuario. Utiliza un ciclo for para controlar la cantidad de repeticiones y muestra el resultado en la consola.

**Proyecto 166:**

```

// Clase que modela un juego de "Tres en Raya"

class TaTeTi {

    val tablero = IntArray(9) // Tablero representado como un arreglo unidimensional

    // Método para imprimir el tablero en formato de matriz 3x3

    fun imprimir() {
        for (fila in 0..2) { // Recorre las filas del tablero
            for (columna in 0..2) // Recorre las columnas del tablero
                print("${this[fila, columna]} ") // Imprime el valor en la posición especificada
            println() // Salto de línea al final de cada fila
        }
        println() // Salto adicional para mayor claridad
    }

    // Sobrecarga del operador []= para asignar un valor en el tablero

    operator fun set(fila: Int, columna: Int, valor: Int) {
        tablero[fila * 3 + columna] = valor // Calcula la posición en el arreglo
        imprimir() // Llama al método imprimir para mostrar el tablero actualizado
    }
}

```

```

// Sobrecarga del operador [] para obtener el valor en una posición del tablero
operator fun get(fila: Int, columna: Int): Int {
    return tablero[fila * 3 + columna] // Calcula y retorna el valor desde el arreglo
}
}

```

// Función principal que simula una partida de "Tres en Raya"

```

fun main(args: Array<String>) {
    val juego = TaTeTi() // Crea una instancia de la clase TaTeTi
    juego[0, 0] = 1 // Marca posición (0, 0) con el valor 1
    juego[0, 2] = 2 // Marca posición (0, 2) con el valor 2
    juego[2, 0] = 1 // Marca posición (2, 0) con el valor 1
    juego[1, 2] = 2 // Marca posición (1, 2) con el valor 2
    juego[1, 0] = 1 // Marca posición (1, 0) con el valor 1
}

```

**Documentación:** Esta clase TaTeTi modela el clásico juego del "Tres en Raya" (Tic-Tac-Toe) utilizando un arreglo unidimensional de tamaño 9 para representar el tablero de juego. Los métodos sobrecargados set y get permiten asignar y recuperar valores en posiciones específicas del tablero usando índices de fila y columna. Además, el método imprimir organiza el tablero en un formato de matriz 3x3 para una visualización clara en la consola. En el método main, se crean movimientos en el tablero utilizando el operador set para simular una partida.

### Proyecto 167:

// Clase que simula diez lanzamientos de dados

```

class Dadoss() {
    val arreglo = IntArray(10) // Arreglo para almacenar los valores de los dados

    // Método para llenar el arreglo con valores aleatorios entre 1 y 6
    fun tirar() {
        for (i in arreglo.indices) // Recorre los índices del arreglo

```

```

        arreglo[i] = ((Math.random() * 6) + 1).toInt() // Genera un número aleatorio entre 1 y 6
    }

    // Sobrecarga del operador invoke para acceder al valor de un dado específico por su índice
    operator fun invoke(nro: Int) = arreglo[nro] // Retorna el valor del arreglo en el índice indicado
}

// Función principal que prueba la funcionalidad de la clase Dadoss
fun main(args: Array<String>) {
    var dados = Dadoss() // Crea una instancia de la clase Dadoss
    dados.tirar() // Llama al método tirar para llenar el arreglo con valores aleatorios

    // Imprime los valores de los dados utilizando el operador invoke directamente
    println(dados(0)) // Muestra el valor del dado en la posición 0
    println(dados(1)) // Muestra el valor del dado en la posición 1

    // Ciclo para recorrer e imprimir los valores de los dados restantes
    for (i in 2..9)
        println(dados(i)) // Muestra el valor del dado en la posición i
}

```

**Documentación:** La clase Dadoss simula el lanzamiento de diez dados, almacenando sus valores en un arreglo de enteros. Incluye el método tirar para generar valores aleatorios entre 1 y 6 en el arreglo y sobrescribe el operador invoke para acceder directamente a los valores de cada dado por su índice. En la función principal, se lanza una instancia de la clase, se generan los valores aleatorios, y se imprimen en la consola utilizando tanto el operador invoke como un ciclo for.

#### **Proyecto 168:**

```

// Clase que representa un vector de enteros con cinco elementos
class Vector4 {
    val arreglo = IntArray(5, { it }) // Inicializa el vector con valores iguales a su índice (0, 1, 2, 3, 4)
}

```

```

// Método para imprimir los valores del vector
fun imprimir() {
    for (elemento in arreglo) // Itera sobre los elementos del vector
        print("$elemento ") // Imprime cada elemento con un espacio
    println() // Agrega un salto de línea al final
}

// Sobrecarga del operador += para realizar la suma elemento a elemento con otro vector
operator fun plusAssign(vec2: Vector4) {
    for (i in arreglo.indices) // Recorre los índices del arreglo
        arreglo[i] += vec2.arreglo[i] // Suma el valor correspondiente del segundo vector al actual
}
}

// Función principal para probar la clase Vector4
fun main(args: Array<String>) {
    val vec1 = Vector4() // Crea el primer vector (0, 1, 2, 3, 4)
    vec1.imprimir() // Imprime los valores iniciales del primer vector

    val vec2 = Vector4() // Crea el segundo vector (0, 1, 2, 3, 4)
    vec2.imprimir() // Imprime los valores iniciales del segundo vector

    vec1 += vec2 // Realiza la suma elemento a elemento entre los vectores
    vec1.imprimir() // Imprime el vector resultante después de la suma
}

```

**Documentación:** La clase Vector4 representa un vector de cinco enteros inicializados con valores iguales a su índice (0, 1, 2, 3, 4). Proporciona el método imprimir para mostrar los valores del vector y sobrecarga el operador += para realizar la suma elemento a elemento entre dos instancias de Vector4. En el programa principal, se crean dos vectores, se imprimen sus valores iniciales, se suma el segundo al primero usando el operador +=, y se imprime el resultado final para verificar la operación.

### Proyecto 169:

// Data class que representa a un alumno con su documento y nombre

```
data class Alumnoo(val documento: Int, val nombre: String)
```

// Clase que representa un curso con un grupo de alumnos

```
class Curso {
```

```
    val alumnos = arrayOf( // Array de alumnos inscritos en el curso
```

```
        Alumnoo(123456, "Marcos"), // Alumno con documento 123456
```

```
        Alumnoo(666666, "Ana"), // Alumno con documento 666666
```

```
        Alumnoo(777777, "Luis") // Alumno con documento 777777
```

```
    )
```

// Sobrecarga del operador 'contains' para verificar si un documento está en la lista de alumnos

```
    operator fun contains(documento: Int): Boolean {
```

```
        return alumnos.any { documento == it.documento } // Devuelve true si el documento coincide con algún alumno
```

```
    }
```

```
}
```

// Función principal que prueba la funcionalidad de la clase Curso

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

```
    val curso1 = Curso() // Crea una instancia del curso
```

// Verifica si el documento 123456 está inscrito en el curso

```
    if (123456 in curso1)
```

```
        println("El alumno Marcos está inscrito en el curso") // Resultado positivo
```

```
    else
```

```
        println("El alumno Marcos no está inscrito en el curso") // Resultado negativo
```

```
}
```

**Documentación:** La clase Curso gestiona un conjunto de alumnos inscritos en un curso, representados mediante una estructura array de objetos Alumno, definidos como una data class con propiedades de documento y nombre. La clase sobrecarga el operador contains, lo que permite verificar de manera sencilla si un documento específico corresponde a algún alumno inscrito. En el programa principal, se crea una instancia de Curso y se utiliza el operador in para verificar si un alumno con un documento particular está inscrito, mostrando el resultado en la consola.

#### Proyecto 170:

// Función que suma una cantidad indefinida de números enteros

```
fun sumar(vararg numeros: Int): Int { // Utiliza 'vararg' para aceptar múltiples argumentos
```

```
    var suma = 0 // Variable para acumular la suma
```

```
    for (elemento in numeros) // Recorre cada número recibido en los argumentos
```

```
        suma += elemento // Suma cada número al acumulador
```

```
    return suma // Retorna la suma total
```

```
}
```

// Función principal que prueba la funcionalidad de la función sumar

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

```
    val total = sumar(10, 20, 30, 40, 50) // Llama a la función con cinco números
```

```
    println(total) // Imprime el resultado de la suma
```

```
}
```

**Documentación:** Este programa incluye una función llamada sumar que utiliza el concepto de parámetros variables (vararg) para sumar una cantidad indefinida de números enteros. La función recorre los valores recibidos utilizando un ciclo for y acumula el total en una variable. En la función principal, se llama a sumar con un conjunto de valores predefinidos y se imprime el resultado en la consola.

#### Proyecto 171:

// Enumeración que define los tipos de operaciones disponibles

```
enum class Operacion {
```

```
    SUMA, // Operación de suma
```

```
    PROMEDIO // Operación de promedio
```

```
}
```



```
// Función que realiza la operación especificada sobre un conjunto de números

fun operar(tipoOperacion: Operacion, vararg arreglo: Int): Int {

    // Selecciona la operación según el valor de tipoOperacion

    when (tipoOperacion) {

        Operacion.SUMA -> return arreglo.sum()      // Retorna la suma de los elementos del arreglo

        Operacion.PROMEDIO -> return arreglo.average().toInt() // Retorna el promedio de los
        elementos como entero

    }

}
```

```
// Función principal que prueba las operaciones definidas

fun main(args: Array<String>) {

    val resultado1 = operar(Operacion.SUMA, 10, 20, 30) // Realiza la suma de 10, 20 y 30

    println("La suma es $resultado1") // Imprime el resultado de la suma


    val resultado2 = operar(Operacion.PROMEDIO, 10, 20, 30) // Calcula el promedio de 10, 20 y
    30

    println("El promedio es $resultado2") // Imprime el resultado del promedio

}
```

**Documentación:** Este programa define una enumeración `Operacion` con dos operaciones: `SUMA` y `PROMEDIO`. Incluye la función `operar`, que utiliza parámetros variables (`vararg`) para aceptar un número indefinido de enteros y realiza la operación especificada, ya sea sumando todos los valores o calculando el promedio. En el método `main`, se realizan ambas operaciones y se imprimen los resultados en la consola.

#### Proyecto 172:

```
// Función que calcula la cantidad de edades mayores o iguales a 18

fun cantidadMayores(vararg edades: Int) =

    edades.count { it >= 18 } // Utiliza el método count para contar los elementos que cumplen la
    condición


// Función principal que prueba la funcionalidad de la función cantidadMayores
```

```

fun main(args: Array<String>) =

    println( // Imprime directamente el resultado de la función cantidadMayores

        cantidadMayores(3, 65, 32, 23, 2, 98, 23, 45, 15) // Llama a la función con un conjunto de edades

    )

```

**Documentación:** Este programa define una función llamada cantidadMayores que utiliza parámetros variables (vararg) para aceptar un número indefinido de edades como entrada. La función utiliza el método count para contar cuántos elementos del arreglo cumplen con la condición de ser mayores o iguales a 18. En la función principal (main), se llama a cantidadMayores con un conjunto de valores de edades y se imprime directamente el resultado en la consola.

### Proyecto 173:

```

fun main(args: Array<String>) {

    // Declaración de una lista de cadenas con los días de la semana

    var lista1: List<String> = listOf(

        "lunes", "martes", "miercoles", "jueves",

        "viernes", "sábado", "domingo"

    )

    // Imprimir la lista completa

    println("Imprimir la lista completa")

    println(lista1) // Muestra todos los elementos de la lista

    // Acceder e imprimir el primer elemento de la lista por índice

    println("Imprimir el primer elemento de la lista")

    println(lista1[0]) // Accede al primer elemento usando índices

    // Acceder e imprimir el primer elemento utilizando el método first

    println("Imprimir el primer elemento de la lista")

    println(lista1.first()) // Usa el método first para obtener el primer elemento

    // Acceder e imprimir el último elemento de la lista utilizando el método last

```

```

println("Imprimir el último elemento de la lista")

println(lista1.last()) // Usa el método last para obtener el último elemento

// Acceder e imprimir el último elemento utilizando el tamaño de la lista
println("Imprimir el último elemento de la lista")
println(lista1[lista1.size - 1]) // Accede al último elemento mediante índices

// Imprimir la cantidad de elementos de la lista
println("Imprimir la cantidad de elementos de la lista")
println(lista1.size) // Muestra el tamaño de la lista

// Recorrer la lista completa utilizando un ciclo for
println("Recorrer la lista completa con un for")
for (elemento in lista1)
    print("$elemento ") // Imprime cada elemento en una línea
println() // Salto de línea al final

// Imprimir cada elemento junto con su posición en la lista
println("Imprimir el elemento y su posición")
for (posicion in lista1.indices)
    print("[${posicion}] ${lista1[posicion]} ") // Muestra el índice y el elemento
}

```

**Documentación:** Este programa maneja una lista de cadenas de texto que representan los días de la semana. Permite realizar diversas operaciones como imprimir la lista completa, acceder al primer y último elemento por índice y mediante métodos de listas (first, last), determinar la cantidad de elementos de la lista, recorrerla completamente utilizando un ciclo for, y mostrar cada elemento junto con su posición. Proporciona ejemplos claros para trabajar con listas en Kotlin.

#### Proyecto 174:

```

// Función que solicita un entero al usuario y lo retorna
fun cargar(): Int {

```

```

    print("Ingrese un entero:") // Muestra un mensaje en la consola

    val valor = readLine()!!.toInt() // Lee la entrada del usuario y la convierte a entero

    return valor // Retorna el valor ingresado
}

// Función principal que utiliza la función cargar para crear una lista de enteros
fun main(args: Array<String>) {
    // Crea una lista de 5 enteros llenada mediante llamadas a la función cargar
    var lista1: List<Int> = List(5, { cargar() })

    // Imprime la lista completa en la consola
    println(lista1) // Muestra los elementos ingresados
}

```

**Documentación:** Este programa define la función cargar, que solicita al usuario que ingrese un número entero y retorna dicho valor. En el método main, se utiliza esta función para inicializar una lista de enteros con cinco elementos mediante el constructor de listas, ejecutando cargar cinco veces para llenar la lista. Finalmente, se imprime la lista completa en la consola, mostrando los valores ingresados por el usuario.

#### **Proyecto 175:**

```

fun main(args: Array<String>) {
    // Crear una lista mutable que almacena edades iniciales
    val edades: MutableList<Int> = mutableListOf(23, 67, 12, 35, 12)

    // Mostrar la lista completa de edades
    println("Lista de edades")
    println(edades) // Imprime los elementos iniciales

    // Modificar el primer elemento de la lista
    edades[0] = 60
    println("Lista completa después de modificar la primer edad")
}

```

```
println(edades) // Imprime la lista después de la modificación

// Acceder e imprimir el primer elemento de la lista
println("Primera edad almacenada en la lista")
println(edades[0]) // Imprime la primera edad

// Calcular y mostrar el promedio de las edades
println("Promedio de edades")
println(edades.average()) // Calcula el promedio con el método average

// Agregar una nueva edad al final de la lista
println("Agregamos una edad al final de la lista:")
edades.add(50) // Agrega el valor 50 al final
println("Lista de edades")
println(edades) // Imprime la lista actualizada

// Agregar una nueva edad al principio de la lista
println("Agregamos una edad al principio de la lista:")
edades.add(0, 17) // Inserta el valor 17 en la posición 0
println("Lista de edades")
println(edades) // Imprime la lista actualizada

// Eliminar la primera edad de la lista
println("Eliminamos la primer edad de la lista:")
edades.removeAt(0) // Elimina el elemento en la posición 0
println("Lista de edades")
println(edades) // Imprime la lista después de la eliminación

// Contar cuántas personas son mayores de edad (18 años o más)
```

```

print("Cantidad de personas mayores de edad:")

val cant = edades.count { it >= 18 } // Cuenta los elementos >= 18

println(cant) // Imprime la cantidad

// Eliminar todas las edades que sean iguales a 12
edades.removeAll { it == 12 } // Elimina todos los elementos con valor 12

println("Lista de edades después de borrar las que tienen 12 años")

println(edades) // Imprime la lista actualizada

// Borrar completamente la lista
edades.clear() // Elimina todos los elementos

println("Lista de edades luego de borrar la lista en forma completa")

println(edades) // Imprime la lista vacía

// Verificar si la lista está vacía
if (edades.isEmpty()) // Comprueba si la lista está vacía
    println("No hay edades almacenadas en la lista") // Mensaje si está vacía
}

```

**Documentación:** Este programa utiliza una lista mutable (`MutableList`) para almacenar edades de personas, ofreciendo diversas operaciones para modificar, consultar y analizar los datos. A lo largo del programa, se imprimen la lista completa, la primera edad almacenada, el promedio de las edades, y la cantidad de personas mayores de edad. Además, se realizan modificaciones como actualizar el primer elemento, agregar edades al principio y al final, eliminar elementos específicos, y borrar completamente la lista. El flujo asegura que cada operación se visualice en la consola para facilitar su comprensión.

#### **Proyecto 176:**

```

fun main(args: Array<String>) {

    // Crear una lista mutable de 20 elementos con valores aleatorios entre 1 y 6

    val lista: MutableList<Int> = MutableList(20) { ((Math.random() * 6) + 1).toInt() }

    // Imprimir la lista completa de valores generados aleatoriamente

```

```

println("Lista completa")

println(lista) // Muestra todos los elementos generados

// Contar la cantidad de elementos con el valor 1 en la lista
val cant = lista.count { it == 1 } // Utiliza el método count para contar las ocurrencias de 1
println("Cantidad de elementos con el valor 1: $cant") // Imprime la cantidad

// Eliminar todos los elementos con el valor 6 de la lista
lista.removeAll { it == 6 } // Usa el método removeAll para eliminar valores iguales a 6
println("Lista luego de borrar los elementos con el valor 6")
println(lista) // Muestra la lista actualizada después de la eliminación
}

```

**Documentación:** Este programa genera una lista mutable (MutableList) con 20 elementos enteros aleatorios, simulando valores entre 1 y 6. Luego, calcula la cantidad de elementos que tienen el valor 1 utilizando el método count y elimina todos los elementos con el valor 6 mediante el método removeAll. Finalmente, imprime la lista inicial, la cantidad de valores iguales a 1, y la lista actualizada después de la eliminación, proporcionando una demostración completa de manipulación de listas en Kotlin.

#### **Proyecto 177:**

```

// Clase que representa una persona con atributos de nombre y edad
class Persona00(var nombre: String, var edad: Int) {

    // Método para imprimir la información de la persona
    fun imprimir() {
        println("Nombre: $nombre y tiene una edad de $edad") // Muestra el nombre y la edad
    }

    // Método para verificar si la persona es mayor de edad
    fun esMayorEdad() {
        if (edad >= 18) // Condición para comprobar si la edad es mayor o igual a 18
            println("Es mayor de edad $nombre") // Mensaje para mayores de edad
    }
}

```

```

        else

            println("No es mayor de edad $nombre") // Mensaje para menores de edad
        }
    }

// Función principal que trabaja con una lista de objetos Persona00
fun main(args: Array<String>) {
    // Crear una lista mutable de objetos Persona00
    val personas: MutableList<Persona00> = mutableListOf(
        Persona00("Juan", 22), // Persona con nombre "Juan" y edad 22
        Persona00("Ana", 19), // Persona con nombre "Ana" y edad 19
        Persona00("Marcos", 12) // Persona con nombre "Marcos" y edad 12
    )

    // Imprimir la información de todas las personas en la lista
    println("Listado de todas personas")
    personas.forEach { it.imprimir() } // Recorre la lista y llama al método imprimir

    // Contar cuántas personas son mayores de edad
    val cant = personas.count { it.edad >= 18 } // Utiliza count para calcular mayores de edad
    println("La cantidad de personas mayores de edad es $cant") // Muestra la cantidad calculada
}

```

**Documentación:** La clase Persona00 representa una persona con atributos de nombre y edad. Ofrece dos métodos: imprimir, para mostrar la información de la persona, y esMayorEdad, que verifica si la persona es mayor de edad basándose en su edad. En la función principal, se crea una lista mutable de personas con diferentes edades, se muestra su información mediante un recorrido con forEach, y se calcula la cantidad de personas mayores de edad utilizando el método count. Esto permite analizar y manipular datos relacionados con un grupo de personas de manera eficiente.

#### Proyecto 178:

```

fun main(args: Array<String>) {

```



```

// Crear una lista inmutable de 100 elementos con valores aleatorios entre 0 y 20
val lista1 = List(100) { ((Math.random() * 21)).toInt() } // Genera los números aleatorios
println(lista1) // Imprime la lista generada

// Variables para contar los elementos en diferentes rangos
var cant1 = 0 // Cantidad de elementos entre 1 y 4
var cant2 = 0 // Cantidad de elementos entre 5 y 8
var cant3 = 0 // Cantidad de elementos entre 10 y 13

// Recorrer la lista y contar los elementos en los rangos definidos
lista1.forEach {
    when (it) {
        in 1..4 -> cant1++ // Incrementa si el valor está entre 1 y 4
        in 5..8 -> cant2++ // Incrementa si el valor está entre 5 y 8
        in 10..13 -> cant3++ // Incrementa si el valor está entre 10 y 13
    }
}

// Imprimir los conteos para cada rango
println("Cantidad de valores comprendidos entre 1..4: $cant1")
println("Cantidad de valores comprendidos entre 5..8: $cant2")
println("Cantidad de valores comprendidos entre 10..13: $cant3")

// Verificar si el valor 20 está presente en la lista
if (lista1.contains(20)) // Usa el método contains para verificar la presencia del 20
    println("La lista contiene el 20") // Mensaje si está presente
else
    println("La lista no contiene el 20") // Mensaje si no está presente
}

```

**Documentación:** Este programa genera una lista inmutable de 100 elementos enteros aleatorios comprendidos entre 0 y 20. Utiliza el método `forEach` para contar cuántos elementos están en los rangos 1-4, 5-8, y 10-13. También verifica si el valor 20 está presente en la lista utilizando el método `contains`. Finalmente, imprime la lista generada, los resultados de los conteos y un mensaje indicando si el valor 20 está o no en la lista.

**Proyecto 179:**

// Clase que representa un empleado con su nombre y sueldo

```
class Empleado0(var nombre: String, var sueldo: Double) {
```

```
    // Método para imprimir la información del empleado
```

```
    fun imprimir() {
```

```
        println("$nombre tiene un sueldo de $sueldo") // Muestra el nombre y sueldo del empleado
```

```
    }
```

```
}
```

// Función para calcular y mostrar el gasto total en sueldos de los empleados

```
fun calcularGastos(empleados: MutableList<Empleado0>) {
```

```
    val suma = empleados.sumByDouble { it.sueldo } // Suma los sueldos de todos los empleados
```

```
    println("Total de gastos de la empresa:$suma") // Muestra el gasto total
```

```
}
```

// Función principal que trabaja con la lista de empleados

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

```
    // Inicializa una lista mutable de empleados con datos iniciales
```

```
    val empleados: MutableList<Empleado0> = mutableListOfOf(
```

```
        Empleado0("Juan", 2000.0), // Empleado con nombre "Juan" y sueldo 2000
```

```
        Empleado0("Ana", 3500.0), // Empleado con nombre "Ana" y sueldo 3500
```

```
        Empleado0("Carlos", 1500.0) // Empleado con nombre "Carlos" y sueldo 1500
```

```
    )
```

```

// Imprime la información de todos los empleados en la lista
empleados.forEach { it.imprimir() }

// Calcula y muestra el gasto total inicial en sueldos
calcularGastos(empleados)

// Agrega un nuevo empleado a la lista
empleados.add(Empleado0("Marcos", 3000.0)) // Empleado con nombre "Marcos" y sueldo 3000

// Imprime el gasto total luego de agregar al nuevo empleado
println("Gastos luego de ingresar un nuevo empleado que cobra 3000")
calcularGastos(empleados)
}

```

**Documentación:** La clase Empleado0 representa a un empleado con atributos como nombre y sueldo, y un método imprimir para mostrar su información. La función calcularGastos recibe una lista mutable de empleados y calcula el total de los sueldos utilizando el método sumByDouble. En la función main, se inicializa una lista mutable de empleados, se muestran sus detalles, y se calcula el gasto total de la empresa antes y después de agregar un nuevo empleado. Esto permite gestionar y analizar los costos asociados al personal de manera eficiente.

#### **Proyecto 180:**

```

// Función que solicita al usuario que ingrese una altura y la retorna como Float
fun cargaaar(): Float {
    print("Ingresa la altura de la persona (Ej. 1.92):") // Solicita al usuario la altura
    val valor = readln().toFloat() // Convierte la entrada a Float
    return valor // Retorna la altura ingresada
}

// Función principal que gestiona la lógica del programa
fun main(args: Array<String>) {
    // Crear una lista inmutable de 5 alturas ingresadas por el usuario

```

```

val lista1: List<Float> = List(5) { cargaaar() } // Llama a la función cargaaar para llenar la lista

// Calcular el promedio de las alturas
val promedio = lista1.average() // Calcula el promedio de los valores en la lista
println("La altura promedio de las personas es $promedio") // Imprime el promedio

// Contar cuántas alturas son mayores y menores al promedio
val altos = lista1.count { it > promedio } // Cuenta las alturas mayores al promedio
val bajos = lista1.count { it < promedio } // Cuenta las alturas menores al promedio

// Imprimir los resultados de los conteos
println("La cantidad de personas más altas al promedio es: $altos")
println("La cantidad de personas más bajas al promedio es: $bajos")
}

```

**Documentación:** Este programa solicita las alturas de cinco personas, las almacena en una lista inmutable (List) de valores tipo Float, y calcula el promedio de las alturas utilizando el método average(). Luego, determina cuántas personas tienen una altura mayor y menor al promedio mediante el método count. Imprime tanto el promedio calculado como las cantidades de personas más altas y más bajas en relación al promedio.

#### **Proyecto 181:**

```

fun main(args: Array<String>) {

    // Declaración de un Map inmutable con pares clave-valor (países y su población)
    val paises: Map<String, Int> = mapOf(
        Pair("argentina", 40000000), // Clave: "argentina", Valor: 40000000
        Pair("españa", 46000000), // Clave: "españa", Valor: 46000000
        Pair("uruguay", 3400000) // Clave: "uruguay", Valor: 3400000
    )

    // Imprimir el mapa completo
    println("Listado completo del Map")
}

```

```

println(paises) // Muestra todos los pares clave-valor

// Recorrer el mapa e imprimir cada clave y su valor
for ((clave, valor) in paises)
    println("Para la clave $clave tenemos almacenado $valor") // Muestra cada entrada

// Mostrar la cantidad de elementos en el mapa
println("La cantidad de elementos del mapa es ${paises.size}")

// Verificar y obtener la población de "argentina"
val cantHabitantes1: Int? = paises["argentina"] // Busca el valor asociado a la clave "argentina"
if (cantHabitantes1 != null)
    println("La cantidad de habitantes de argentina es $cantHabitantes1") // Si existe, muestra el
valor

// Verificar la población de "brasil" (no existe en el mapa)
val cantHabitantes2: Int? = paises["brasil"] // Busca la clave "brasil"
if (cantHabitantes2 != null)
    println("La cantidad de habitantes de brasil es $cantHabitantes2") // Si existe, muestra el valor
else
    println("brasil no se encuentra cargado en el Map") // Mensaje si no existe la clave

// Calcular la suma de la población de todos los países
var suma = 0 // Variable acumuladora
paises.forEach { suma += it.value } // Recorre el mapa y acumula los valores (población)
println("Cantidad total de habitantes de todos los paises es $suma") // Imprime el total
}

```

**Documentación:** Este programa utiliza un Map para almacenar pares de clave-valor, donde la clave representa el nombre de un país y el valor la cantidad de habitantes de dicho país. Permite imprimir el mapa completo, recorrerlo mostrando cada clave y su valor, contar el total de elementos, verificar

si un país específico está en el mapa, y sumar la cantidad total de habitantes de todos los países. También maneja casos en los que una clave buscada no existe en el mapa.

### **Proyecto 182:**

```
// Función que imprime cada producto y su precio
fun imprimir(productos: Map<String, Float>) {
    for ((clave, valor) in productos) // Recorre el mapa clave-valor
        println("$clave tiene un precio $valor") // Imprime cada producto con su precio
    println() // Salto de línea adicional para separar bloques de salida
}

// Función que cuenta y muestra la cantidad de productos con precio superior a 20
fun cantidadPrecioMayor20(productos: Map<String, Float>) {
    val cant = productos.count { it.value > 20 } // Cuenta los elementos con valor > 20
    println("Cantidad de productos con un precio superior a 20: $cant") // Imprime el resultado
}

// Función principal que gestiona el flujo del programa
fun main(args: Array<String>) {
    // Crear un mapa inmutable con productos y precios
    val productos: Map<String, Float> = mapOf(
        "papas" to 12.5f,    // Producto: papas, Precio: 12.5
        "manzanas" to 26f,   // Producto: manzanas, Precio: 26
        "peras" to 31f,      // Producto: peras, Precio: 31
        "mandarinas" to 15f, // Producto: mandarinas, Precio: 15
        "pomelos" to 28f     // Producto: pomelos, Precio: 28
    )

    imprimir(productos) // Llama a la función para imprimir productos y precios
    cantidadPrecioMayor20(productos) // Llama a la función para contar precios mayores a 20
}
```

```
}
```

**Documentación:** Este programa utiliza un Map para almacenar productos con su respectivo precio, donde la clave es el nombre del producto y el valor es su precio en tipo Float. Incluye dos funciones: imprimir, que recorre el mapa y muestra cada producto junto con su precio, y cantidadPrecioMayor20, que calcula cuántos productos tienen un precio superior a 20. En la función principal (main), se inicializa el mapa con cinco productos, se llama a ambas funciones para visualizar los datos y analizar los precios.

### Proyecto 183:

```
// Función para cargar palabras en el diccionario
```

```
fun cargar(diccionario: MutableMap<String, String>) {  
    do {  
        print("Ingrese palabra en castellano:") // Solicita la palabra en español  
        val palCastellano = readLine()!! // Lee la palabra ingresada  
        print("Ingrese palabra en ingles:") // Solicita la palabra en inglés  
        val palIngles = readLine()!! // Lee la palabra ingresada  
        diccionario[palIngles] = palCastellano // Agrega la palabra al diccionario (clave: inglés, valor: español)  
        print("Continúa cargando otra palabra en el diccionario? (si/no):") // Pregunta si desea continuar  
        val fin = readLine()!! // Lee la respuesta  
    } while (fin == "si") // Repite mientras la respuesta sea "si"  
}
```

```
// Función para listar todas las palabras y sus traducciones en el diccionario
```

```
fun listado(diccionario: MutableMap<String, String>) {  
    for ((ingles, castellano) in diccionario) // Recorre las entradas del diccionario  
        println("$ingles: $castellano") // Imprime cada par inglés-español  
    println() // Salto de línea para separar bloques de salida  
}
```

```
// Función para consultar la traducción de una palabra en inglés
```

```
fun consultaIngles(diccionario: MutableMap<String, String>) {
```

```

print("Ingrese una palabra en ingles para verificar su traducción:") // Solicita una palabra en inglés
val ingles = readLine() // Lee la palabra ingresada
if (ingles in diccionario) // Verifica si la palabra existe en el diccionario
    println("La traducción en castellano es ${diccionario[ingles]}") // Muestra la traducción si existe
else
    println("No existe esa palabra en el diccionario") // Muestra un mensaje si no está en el
    diccionario
}

```

```

// Función principal que organiza el flujo del programa
fun main(args: Array<String>) {
    val diccionario: MutableMap<String, String> = mutableMapOf() // Inicializa un diccionario
    mutable vacío
    cargar(diccionario) // Llama a la función para cargar palabras en el diccionario
    listado(diccionario) // Llama a la función para listar todas las palabras y traducciones
    consultaIngles(diccionario) // Llama a la función para consultar una traducción específica
}

```

**Documentación:** Este programa permite gestionar un diccionario bilingüe (inglés-español) utilizando un MutableMap. Incluye funciones para cargar palabras en el diccionario, listar todas las palabras con sus traducciones, y consultar la traducción de una palabra ingresada en inglés. El programa interactúa con el usuario mediante entradas de teclado y muestra resultados en la consola.

#### **Proyecto 184:**

```

// Data class que representa un producto con nombre, precio y cantidad en stock
data class Producto(val nombre: String, val precio: Float, val cantidad: Int)

```

```

// Función para cargar productos iniciales en el mapa
fun cargar(productos: MutableMap<Int, Producto>) {
    productos[1] = Producto("Papas", 13.15f, 200) // Código 1: Papas con precio 13.15 y stock 200
    productos[15] = Producto("Manzanas", 20f, 0) // Código 15: Manzanas con precio 20 y stock 0
    productos[20] = Producto("Peras", 3.50f, 0) // Código 20: Peras con precio 3.50 y stock 0
}

```



```
}
```

```
// Función para listar todos los productos en el mapa
```

```
fun listadoCompleto(productos: MutableMap<Int, Producto>) {  
    println("Listado completo de productos")  
    for ((codigo, producto) in productos) { // Itera por cada par clave-valor en el mapa  
        println("Código: $codigo Descripción: ${producto.nombre} Precio: ${producto.precio} Stock  
Actual: ${producto.cantidad}")  
    }  
    println() // Salto de línea para separar secciones  
}
```

```
// Función para consultar un producto por su código
```

```
fun consultaProducto(productos: MutableMap<Int, Producto>) {  
    print("Ingrese el código de un producto:") // Solicita al usuario el código  
    val codigo = readLine()!!.toInt() // Lee el código como entero  
    if (codigo in productos) { // Verifica si el código está en el mapa  
        println("Nombre: ${productos[codigo]?.nombre} Precio: ${productos[codigo]?.precio} Stock:  
${productos[codigo]?.cantidad}")  
    } else {  
        println("No existe un producto con dicho código") // Mensaje si no está  
    }  
}
```

```
// Función para contar cuántos productos tienen stock igual a 0
```

```
fun sinStock(productos: MutableMap<Int, Producto>) {  
    val cant = productos.count { it.value.cantidad == 0 } // Cuenta productos con stock 0  
    println("Cantidad de artículos sin stock: $cant") // Imprime el resultado  
}
```

```
// Función principal que organiza y ejecuta las funcionalidades del programa
fun main(args: Array<String>) {
    val productos: MutableMap<Int, Producto> = mutableMapOf() // Mapa mutable para almacenar productos

    cargar(productos) // Carga los productos iniciales

    listadoCompleto(productos) // Muestra el listado completo de productos

    consultaProducto(productos) // Permite consultar un producto por su código

    sinStock(productos) // Muestra la cantidad de productos sin stock
}
```

**Documentación:** Este programa administra un inventario de productos utilizando un MutableMap, donde las claves son los códigos de los productos y los valores son instancias de la clase Producto. La clase Producto es una data class que incluye las propiedades nombre, precio y cantidad (stock).

#### **Proyecto 185:**

```
// Data class que representa una materia con su nombre y nota
data class Materia(val nombre: String, val nota: Int)

// Función para cargar alumnos y sus materias en el mapa
fun cargar(alumnos: MutableMap<Int, MutableList<Materia>>()) {
    print("Cuántos alumnos cargará ?:") // Solicita la cantidad de alumnos
    val cant = readLine()!!.toInt() // Lee la cantidad ingresada
    for (i in 1..cant) { // Itera para ingresar datos de cada alumno
        print("Ingrese dni:") // Solicita el DNI del alumno
        val dni = readLine()!!.toInt() // Lee el DNI ingresado
        val listaMaterias = mutableListOf<Materia>() // Lista mutable para las materias del alumno
        do {
            print("Ingrese materia del alumno:") // Solicita el nombre de la materia
            val nombre = readLine()!! // Lee el nombre ingresado
            print("Ingrese nota:") // Solicita la nota de la materia
            val nota = readLine()!!.toInt() // Lee la nota ingresada
```

```

        listaMaterias.add(Materia(nombre, nota)) // Agrega la materia a la lista
        print("Ingrese otra nota (si/no):") // Pregunta si desea ingresar otra materia
        val opcion = readLine()!! // Lee la respuesta
    } while (opcion == "si") // Continúa mientras la respuesta sea "si"
    alumnos[dni] = listaMaterias // Asocia la lista de materias al DNI en el mapa
}
}

```

```

// Función para imprimir la información completa de los alumnos
fun imprimir(alumnos: MutableMap<Int, MutableList<Materia>>) {
    for ((dni, listaMaterias) in alumnos) { // Recorre el mapa
        println("Dni del alumno: $dni") // Imprime el DNI del alumno
        for (materia in listaMaterias) { // Itera sobre la lista de materias
            println("Materia: ${materia.nombre}") // Imprime el nombre de la materia
            println("Nota: ${materia.nota}") // Imprime la nota de la materia
        }
        println() // Salto de línea para separar datos de alumnos
    }
}

```

```

// Función para consultar las materias de un alumno por su DNI
fun consultaPorDni(alumnos: MutableMap<Int, MutableList<Materia>>) {
    print("Ingrese el dni del alumno a consultar:") // Solicita el DNI del alumno
    val dni = readLine()!!.toInt() // Lee el DNI ingresado
    if (dni in alumnos) { // Verifica si el DNI está en el mapa
        println("Cursa las siguientes materias") // Muestra las materias
        val lista = alumnos[dni] // Obtiene la lista de materias asociada al DNI
        if (lista != null) { // Verifica que la lista no sea nula
            for ((nombre, nota) in lista) { // Itera sobre la lista de materias

```

```

        println("Nombre de materia: $nombre") // Imprime el nombre de la materia
        println("Nota: $nota") // Imprime la nota de la materia
    }
}
} else {
    println("No existe un alumno con ese DNI") // Mensaje si el DNI no está en el mapa
}
}

// Función principal que organiza y ejecuta el flujo del programa
fun main(args: Array<String>) {
    val alumnos: MutableMap<Int, MutableList<Materia>> = mutableMapOf() // Inicializa un mapa mutable vacío

    cargar(alumnos) // Llama a la función para cargar los datos
    imprimir(alumnos) // Llama a la función para imprimir los datos
    consultaPorDni(alumnos) // Llama a la función para consultar un alumno por su DNI
}

```

**Documentación:** Este programa gestiona información académica de alumnos mediante un `MutableMap`, donde la clave es el DNI del alumno y el valor es una lista mutable de objetos `Materia`. La clase `Materia`, definida como una data class, almacena el nombre de la materia y la nota obtenida.

#### **Proyecto 186:**

```

// Data class que representa una fecha con día, mes y año
data class Fecha(val dia: Int, val mes: Int, val año: Int)

// Función para cargar fechas y actividades en la agenda
fun cargar(agenda: MutableMap<Fecha, String>) {
    do {
        println("Ingrese fecha")
        print("Ingrese el día:") // Solicita el día
        val dia = readln().toInt() // Lee el día como entero
    }
}

```

```

    print("Ingrese el mes:") // Solicita el mes

    val mes = readln().toInt() // Lee el mes como entero

    print("Ingrese el año:") // Solicita el año

    val año = readln().toInt() // Lee el año como entero

    print("Ingrese todas las actividades para ese día:") // Solicita las actividades

    val actividades = readln() // Lee las actividades como texto

    agenda[Fecha(día, mes, año)] = actividades // Almacena las actividades en la agenda usando la
    fecha como clave

    print("Ingrese otra fecha (si/no):") // Pregunta si desea continuar ingresando fechas

    val opcion = readln() // Lee la opción del usuario

    } while (opcion == "si") // Repite mientras la opción sea "si"
}

// Función para imprimir todas las fechas y actividades de la agenda
fun imprimir(agenda: MutableMap<Fecha, String>) {
    for ((fecha, actividad) in agenda) { // Recorre el mapa de la agenda

        println("Fecha  ${fecha.día}/${fecha.mes}/${fecha.año}") // Imprime la fecha en formato
        DD/MM/AAAA

        println("Actividades: $actividad") // Imprime las actividades para esa fecha

        println() // Salto de línea para separar los registros

    }
}

// Función para consultar las actividades de una fecha específica
fun consultaFecha(agenda: MutableMap<Fecha, String>) {

    println("Ingrese una fecha a consultar")

    print("Ingrese el día:") // Solicita el día

    val día = readln().toInt() // Lee el día como entero

    print("Ingrese el mes:") // Solicita el mes

```

```

val mes = readln().toInt() // Lee el mes como entero

print("Ingrese el año:") // Solicita el año

val año = readln().toInt() // Lee el año como entero

val fecha = Fecha(dia, mes, año) // Crea un objeto Fecha con los valores ingresados

if (fecha in agenda) { // Verifica si la fecha está en la agenda
    println("Actividades: ${agenda[fecha]}") // Imprime las actividades correspondientes
} else {
    println("No existen actividades registradas para dicha fecha") // Mensaje si no hay actividades
}
}

// Función principal que organiza el flujo del programa
fun main(args: Array<String>) {
    val agenda: MutableMap<Fecha, String> = mutableMapOf() // Inicializa una agenda vacía
    cargar(agenda) // Llama a la función para cargar actividades
    imprimir(agenda) // Llama a la función para imprimir la agenda completa
    consultaFecha(agenda) // Llama a la función para consultar actividades por fecha
}

```

**Documentación:** Este programa administra una agenda personal utilizando un `MutableMap`, donde la clave es un objeto `Fecha` (definido como una `data class` con día, mes y año) y el valor es una cadena que contiene las actividades para esa fecha.

**Proyecto 187:**

```

fun main(args: Array<String>) {
    // Declaración de un conjunto mutable con algunos valores iniciales
    val conjunto1: MutableSet<Int> = mutableSetOf(2, 7, 20, 150, 3)

    // Imprimir el listado completo del conjunto
    println("Listado completo del conjunto")

    for (elemento in conjunto1) // Recorre los elementos del conjunto

```

```
    print("$elemento ") // Imprime cada elemento separado por un espacio
println() // Salto de línea para finalizar

// Imprimir la cantidad de elementos en el conjunto
println("Cantidad de elementos del conjunto: ${conjunto1.size}")

// Agregar un nuevo elemento al conjunto
conjunto1.add(500) // Agrega el número 500
println("Listado completo del conjunto luego de agregar el 500")
for (elemento in conjunto1) // Recorre nuevamente el conjunto
    print("$elemento ") // Imprime los elementos del conjunto actualizado
println()

// Intentar agregar nuevamente el número 500 al conjunto
conjunto1.add(500) // No se agrega porque ya existe
println("Listado completo del conjunto luego de volver a agregar el 500")
for (elemento in conjunto1) // Recorre el conjunto
    print("$elemento ") // Imprime los elementos (sin duplicados)
println()

// Verificar si el número 500 está en el conjunto
if (500 in conjunto1)
    println("El 500 está almacenado en el conjunto") // Mensaje si existe
else
    println("El 500 no está almacenado en el conjunto") // Mensaje si no existe

// Eliminar el número 500 del conjunto
println("Eliminamos el elemento 500 del conjunto")
conjunto1.remove(500) // Elimina el número 500
```

```

if (500 in conjunto1)

    println("El 500 está almacenado en el conjunto") // Mensaje si existe después de intentar eliminar
else

    println("El 500 no está almacenado en el conjunto") // Mensaje si se eliminó correctamente


// Contar cuántos elementos tienen valores superiores o iguales a 10
val cant = conjunto1.count { it >= 10 } // Filtra elementos >= 10 y cuenta
println("Cantidad de elementos con valores superiores o igual a 10: $cant") // Imprime el resultado
}

```

**Documentación:** Este programa utiliza un MutableSet para trabajar con un conjunto de valores enteros, mostrando diversas operaciones como agregar, eliminar y contar elementos. Los conjuntos en Kotlin son estructuras que no permiten duplicados, lo que significa que cada valor es único. El programa demuestra cómo usar las propiedades y métodos de un conjunto mutable para realizar manipulaciones y consultas.

#### **Proyecto 188:**

```

// Data class que representa una fecha con día, mes y año
data class Fechaa(val dia: Int, val mes: Int, val año: Int)


// Función principal donde se ejecuta el flujo del programa
fun main(args: Array<String>) {

    // Declaración de un conjunto inmutable que almacena las fechas de los feriados
    var feriados: Set<Fechaa> = setOf(

        Fechaa(1, 1, 2017),    // Año Nuevo
        Fechaa(25, 12, 2017)  // Navidad
    )


    // Solicitar al usuario que ingrese una fecha
    println("Ingrese una fecha")

    print("Ingrese el día:") // Solicita el día

    val dia = readLine()!!.toInt() // Lee el día como entero

```



```

print("Ingrese el mes:") // Solicita el mes

val mes = readLine()!!.toInt() // Lee el mes como entero

print("Ingrese el año:") // Solicita el año

val año = readLine()!!.toInt() // Lee el año como entero


// Verificar si la fecha ingresada está en el conjunto de feriados
if (Fechaa(dia, mes, año) in feriados) {

    println("La fecha ingresada es feriado") // Mensaje si la fecha es un feriado

} else {

    println("La fecha ingresada no es feriado") // Mensaje si no es feriado

}

}

```

**Documentación:** Este programa utiliza un conjunto inmutable (Set) para almacenar fechas de feriados representadas con la clase Fechaa. A través de una entrada del usuario, permite verificar si una fecha ingresada corresponde a un día feriado registrado en el conjunto.

#### **Proyecto 189:**

// Función para generar un cartón con 6 números únicos entre 1 y 50

```

fun generarCarton(carton: MutableSet<Int>) {

    do {

        val valor = ((Math.random() * 50) + 1).toInt() // Genera un número aleatorio entre 1 y 50

        carton.add(valor) // Agrega el número al conjunto

    } while (carton.size != 6) // Repite hasta que el conjunto tenga exactamente 6 números

}

```

// Función para generar el bolillero con 50 números únicos entre 1 y 50

```

fun generarBolillero(bolillero: MutableSet<Int>) {

    do {

        val valor = ((Math.random() * 50) + 1).toInt() // Genera un número aleatorio entre 1 y 50

        bolillero.add(valor) // Agrega el número al conjunto

    }

```

```

    } while (bolillero.size != 50) // Repite hasta que el conjunto tenga exactamente 50 números
}

// Función para verificar cuántos intentos son necesarios para completar el cartón
fun verificarTriunfo(carton: MutableSet<Int>, bolillero: MutableSet<Int>) {
    var aciertos = 0 // Contador de aciertos en el cartón
    var cantBolillas = 0 // Contador de bolillas extraídas
    var intentos = 0 // Número de intentos necesarios para completar el cartón

    for (bolilla in bolillero) { // Recorre las bolillas del bolillero
        cantBolillas++ // Incrementa el contador de bolillas extraídas
        if (bolilla in carton) { // Verifica si la bolilla está en el cartón
            aciertos++ // Incrementa el contador de aciertos
            if (aciertos == 6) { // Comprueba si el cartón tiene los 6 aciertos
                intentos = cantBolillas // Registra el número de intentos
                break // Termina el bucle una vez que el cartón está completo
            }
        }
    }

    println("Se sacaron $intentos bolillas hasta que el cartón ganó.") // Imprime los intentos necesarios
}

// Función principal donde se ejecuta el programa
fun main(args: Array<String>) {
    val carton: MutableSet<Int> = mutableSetOf() // Inicializa un conjunto mutable para el cartón
    generarCarton(carton) // Genera los números del cartón
    println("Carton de lotería generado")
    println(carton) // Imprime el cartón generado
}

```

```

val bolillero: MutableSet<Int> = mutableSetOf() // Inicializa un conjunto mutable para el bolillero
generarBolillero(bolillero) // Genera los números del bolillero
println("Numeros del bolillero")
println(bolillero) // Imprime los números del bolillero

verificarTriunfo(carton, bolillero) // Verifica los intentos necesarios para completar el cartón
}

```

**Documentación:** Este programa simula una lotería utilizando conjuntos (MutableSet) para manejar tanto el cartón del jugador como las bolillas del bolillero.

### Proyecto 190:

// Función que suma dos números enteros

```

fun sumar(a: Int, b: Int): Int {
    return a + b // Retorna la suma de los dos números
}

```

// Función que resta dos números enteros

```

fun restar(a: Int, b: Int): Int {
    return a - b // Retorna la resta del primer número menos el segundo
}

```

// Función principal donde se ejecuta el programa

```

fun main(args: Array<String>) {
    // Llama a la función sumar con los valores 5 y 7, y almacena el resultado en la variable 'su'
    val su = sumar(5, 7)
    println("La suma es $su") // Imprime el resultado de la suma

    // Llama a la función restar con los valores 10 y 3, y almacena el resultado en la variable 're'
    val re = restar(10, 3)
    println("La resta es $re") // Imprime el resultado de la resta
}

```

}

**Documentación:** Este programa realiza operaciones básicas, como sumar y restar dos números enteros, utilizando funciones definidas por el usuario. La estructura principal consta de dos funciones (sumar y restar) que realizan las operaciones respectivas y retornan el resultado. En la función main, se utilizan estas funciones y se imprimen los resultados en la consola. Este programa es un ejemplo simple y funcional para aprender sobre la definición de funciones y su invocación en Kotlin.