Notas de Kotlin:

2. **Conozca los Operadores y los Tipos en Kotlin**

### **Paso 1: Explorando los Operadores Numéricos**

En esta parte, aprenderemos sobre los operadores matemáticos en Kotlin:

* **Suma (+)**
* **Resta (-)**
* **Multiplicación (\*)**
* **División (/)**

Además, Kotlin admite diferentes tipos de números, como Int, Long, Double y Float.

Aquí tienes algunos ejemplos y sus resultados:

1. 1 + 1 ⇒ 2
2. 53 - 3 ⇒ 50
3. 50 / 10 ⇒ 5
4. 1.0 / 2.0 ⇒ 0.5
5. 2.0 \* 3.5 ⇒ 7.0

Recuerda que los resultados mantienen los tipos de los operandos. Por ejemplo, 1 / 2 da como resultado 0, mientras que 1.0 / 2.0 da como resultado 0.5.

### **Paso 2: Practicando con Tipos**

Kotlin no permite conversiones automáticas entre tipos. Por lo tanto, no puedes asignar directamente un valor de tipo Byte a una variable de tipo Int o String. Sin embargo, puedes realizar conversiones explícitas:

1. Define una variable de tipo Int: val i: Int = 6.
2. Crea una nueva variable y convierte i a tipo Byte: val b1 = i.toByte().
3. Imprime el valor de b1, que será 6.

Además, puedes llamar a métodos en números como si fueran objetos:

* 2.times(3) ⇒ 6
* 3.5.plus(4) ⇒ 7.5
* 2.4.div(2) ⇒ 1.2

Finalmente, para hacer que las constantes numéricas largas sean más legibles, puedes usar guiones bajos en los números:

* val oneMillion = 1\_000\_000
* val socialSecurityNumber = 999\_999\_999L
* val hexBytes = 0xFF\_EC\_DE\_5E
* val bytes = 0b11010010\_01101001\_10010100\_10010010

3. **Compara Condiciones y Booleanos en Kotlin**

En esta tarea, exploraremos los conceptos de booleanos y verificación de condiciones en el lenguaje de programación Kotlin. Al igual que en otros lenguajes, Kotlin utiliza operadores booleanos como menor que, igual a, mayor que, etc. (,, ,, ,, ,, !=, <=, >=).

### **1. Verificación de Proporciones**

Supongamos que tenemos un acuario con peces y plantas. Queremos verificar si la proporción entre el número de peces y el número de plantas es saludable. Aquí está el código:

Kotlin

val numberOfFish = 50

val numberOfPlants = 23

if (numberOfFish > numberOfPlants) {

println("Buena proporción")

} else {

println("Proporción no saludable")

}

Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

Si hay más peces que plantas, muestra “Buena proporción”. De lo contrario, muestra “Proporción no saludable”.

### **2. Utilizando Rangos**

Kotlin nos permite definir fácilmente rangos. Por ejemplo, podemos verificar si el número de peces está dentro del rango de 1 a 100:

Kotlin

val fish = 50

if (fish in 1..100) {

println(fish)

}

Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

En este caso, muestra el número de peces (que es 50).

### **3. Múltiples Casos**

Para condiciones más complejas, podemos usar múltiples casos con if, else if y else. Por ejemplo:

Kotlin

if (numberOfFish == 0) {

println("Acuario vacío")

} else if (numberOfFish < 40) {

println("¡Hay peces!")

} else {

println("¡Eso es un montón de peces!")

}

Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

Si no hay peces, muestra “Acuario vacío”. Si hay menos de 40 peces, muestra “¡Hay peces!”. De lo contrario, muestra “¡Eso es un montón de peces!”.

### **4. Declaración con Rangos**

Podemos simplificar aún más usando una declaración con rangos:

Kotlin

when (numberOfFish) {

0 -> println("Acuario vacío")

in 1..39 -> println("¡Hay peces!")

else -> println("¡Eso es un montón de peces!")

}

Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

Este código hace lo mismo que el ejemplo anterior. Utiliza when para evaluar diferentes casos según el número de peces.

4. **Más Información sobre la Nulabilidad en Kotlin**

En esta tarea, exploraremos las diferencias entre variables que pueden aceptar valores nulos y las que no. Los errores relacionados con valores nulos han sido una fuente común de problemas en la programación. Kotlin busca reducir estos errores al introducir variables que no pueden contener valores nulos.

### **Paso 1: Entendiendo la Nulabilidad**

Por defecto, las variables en Kotlin no pueden contener el valor null. Por ejemplo:

Kotlin

var rocks: Int = null

// Error: null no puede ser un valor de un tipo no nulo Int

Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

Para permitir que una variable pueda ser null, utilizamos el operador de signo de interrogación (?). Por ejemplo:

Kotlin

var marbles: Int? = null

Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

En el caso de tipos de datos más complejos, como listas:

* Podemos permitir que los elementos de la lista sean nulos.
* Podemos permitir que la lista sea nula, pero si no lo es, sus elementos no pueden ser nulos.
* También podemos permitir que tanto la lista como sus elementos sean nulos.

### **Paso 2: Operadores ? y ?:**

Kotlin ofrece atajos para trabajar con valores nulos. Veamos algunos ejemplos:

1. Operador ?. (Safe Call):
   * Nos permite llamar a métodos o acceder a propiedades de una variable que podría ser null.
   * Si la variable es null, la evaluación se detiene y no se llama al método.
   * Ejemplo:
   * Kotlin

var fishFoodTreats = 6

fishFoodTreats = fishFoodTreats?.dec()

* + Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

1. Operador ?: (Elvis Operator):
   * Es una abreviatura para “si no es null, úsalo; de lo contrario, usa un valor predeterminado”.
   * Ejemplo:
   * Kotlin

fishFoodTreats = fishFoodTreats?.dec() ?: 0

* + Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).
  + Si fishFoodTreats es null, se asigna el valor 0.

1. Operador de Aserción No Nulo (Double-Bang):
   * Convierte cualquier valor en un tipo no nulo y produce una excepción si el valor es null.
   * Ejemplo:
   * Kotlin

val len = s!!.length

// Lanza una excepción NullPointerException si s es null

* + Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

Recuerda que el operador de aserción no nulo a veces se llama “operador Elvis” debido a su apariencia similar al peinado de copete de Elvis Presley.

5. **Explorando Matrices, Listas y Bucles en Kotlin**

### **Paso 1: Listas y Matrices**

1. Listas:
   * Las listas son similares a las listas en otros lenguajes.
   * Puedes declarar una lista inmutable con listOf.
   * Ejemplo:
   * Kotlin

val school = listOf("mackerel", "trout", "halibut")

println(school)

// Salida: [mackerel, trout, halibut]

* + Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).
  + También puedes declarar una lista mutable con mutableListOf.

1. Matrices:
   * Las matrices son similares a las listas, pero su tamaño es fijo.
   * No puedes agregar ni quitar elementos después de crear una matriz.
   * Puedes declarar una matriz con arrayOf.
   * Ejemplo:
   * Kotlin

val numbers = intArrayOf(1, 2, 3)

println(java.util.Arrays.toString(numbers))

// Salida: [1, 2, 3]

* + Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).
  + También puedes mezclar diferentes tipos en una matriz.

1. Operador + para Combinar Matrices:
   * Puedes combinar dos matrices con el operador +.
   * Ejemplo:
   * Kotlin

val numbers = intArrayOf(1, 2, 3)

val numbers3 = intArrayOf(4, 5, 6)

val combinedArray = numbers3 + numbers

println(combinedArray[5])

// Salida: 3

* + Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

### **Paso 2: Creando Bucles**

1. Bucle for:
   * Puedes usar un bucle for para iterar a través de los elementos de una lista o matriz.
   * Ejemplo:
   * Kotlin

val school = arrayOf("shark", "salmon", "minnow")

for (element in school) {

print(element + " ")

}

// Salida: shark salmon minnow

* + Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

1. Recorriendo Índices y Elementos al Mismo Tiempo:
   * Puedes recorrer los elementos y sus índices al mismo tiempo.
   * Ejemplo:
   * Kotlin

for ((index, element) in school.withIndex()) {

println("Elemento $index: $element")

}

// Salida:

// Elemento 0: shark

// Elemento 1: salmon

// Elemento 2: minnow

* + Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

1. Inicialización de Matrices con Código:
   * Puedes inicializar matrices con código en lugar de asignar valores manualmente.
   * Ejemplo:
   * Kotlin

val array = Array(5) { it \* 2 }

println(java.util.Arrays.toString(array))

// Salida: [0, 2, 4, 6, 8]

* + Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

Lección 2 Funciones:

## **Explorando la Función** main() **en Kotlin**

### **Paso 1: Creando un Archivo de Kotlin**

1. Abre IntelliJ IDEA.
2. En la ventana del proyecto, busca la carpeta src dentro del proyecto “Hello Kotlin” (según la versión de tu IDE, es posible que debas buscar dentro de una carpeta adicional).
3. Haz clic con el botón derecho en la carpeta src y selecciona “Nuevo > Archivo de Kotlin”.
4. Asigna al archivo el nombre “Hola”.
5. Haz clic en “Aceptar”.

Ahora tienes un archivo llamado “Hola.kt” en la carpeta src.

### **Paso 2: Agregando Código y Ejecutando tu Programa**

1. Escribe o pega el siguiente código en el archivo “Hola.kt”:
2. Kotlin

fun main(args: Array<String>) {

println("¡Hola, mundo!")

}

1. Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).
2. Para ejecutar el programa, haz clic en el triángulo verde situado a la izquierda de la función main(). Luego selecciona “Ejecutar ‘HelloKt’” en el menú.

IntelliJ IDEA compilará el programa y lo ejecutará. Los resultados aparecerán en el registro de la parte inferior.

### **Paso 3: Pasando Argumentos a** main()

Dado que estás ejecutando el programa desde IntelliJ IDEA y no desde la línea de comandos, debes especificar los argumentos del programa de forma un poco diferente:

1. Selecciona “Ejecutar > Editar configuraciones”. Se abrirá la ventana de configuraciones de ejecución/depuración.
2. Escribe “Kotlin!” en el campo “Argumentos del programa”.
3. Haz clic en “Aceptar”.

### **Paso 4: Cambiando el Código para Usar una Plantilla de Cadena**

Una plantilla de cadena te permite insertar una variable o expresión en una cadena. Las llaves enmarcan la expresión. Por ejemplo:

Kotlin

fun main(args: Array<String>) {

val name = args[0]

println("¡Hola, $name!")

}

Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

En este caso, el programa tomará el primer argumento pasado al programa y lo usará en lugar de “mundo”. Por ejemplo, si ejecutas el programa con el argumento “Kotlin”, mostrará “¡Hola, Kotlin!”.

2.

## **Por qué (casi) todo tiene un valor en Kotlin**

En Kotlin, a diferencia de algunos otros lenguajes de programación, casi todo es una expresión y tiene un valor. Esto significa que incluso las instrucciones que normalmente no devuelven un valor, como las declaraciones if, tienen un valor en Kotlin.

### **1. Ejemplo con la función** println()

En el archivo “Hello.kt”, tenemos la siguiente función:

Kotlin

fun main(args: Array<String>) {

val isUnit = println("This is an expression")

println(isUnit)

}

Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

* La función println() muestra la cadena “This is an expression” y devuelve un valor especial llamado kotlin.Unit.
* El valor de isUnit se asigna a kotlin.Unit.
* Al imprimir isUnit, obtenemos la salida: kotlin.Unit.

### **2. Ejemplo con una expresión condicional**

En el mismo archivo “Hello.kt”, declaramos una variable temperature con un valor de 10. Luego, creamos otra variable isHot utilizando una expresión condicional:

Kotlin

val temperature = 10

val isHot = if (temperature > 50) true else false

println(isHot)

Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

* La expresión if (temperature > 50) true else false evalúa si la temperatura es mayor que 50.
* El valor de isHot se asigna a false.
* Al imprimir isHot, obtenemos la salida: false.

### **3. Utilizando valores de expresiones en plantillas de cadena**

También podemos utilizar el valor de una expresión en una plantilla de cadena. Por ejemplo:

Kotlin

val message = "The water temperature is ${if (temperature > 50) "too warm" else "OK"}."

println(message)

Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

* La plantilla de cadena utiliza la expresión condicional para determinar si la temperatura es “too warm” o “OK”.
* Al imprimir message, obtenemos la salida: The water temperature is OK.

En resumen, en Kotlin, casi todo tiene un valor, lo que nos permite escribir código más conciso y expresivo. Sin embargo, los bucles son una excepción a esta regla, ya que no tienen un valor razonable y, por lo tanto, no se pueden asignar directamente a una variable.

3.

## **Aprendiendo más sobre funciones y la expresión** when **en Kotlin**

### **Paso 1: Creando algunas funciones**

En este paso, vamos a poner en práctica lo que hemos aprendido y crear funciones con diferentes tipos. Puedes reemplazar el contenido de “Hello.kt” con el siguiente código:

1. Escribe una función llamada feedTheFish() que llame a randomDay() para obtener un día de la semana al azar. Utiliza una plantilla de cadena para imprimir un mensaje sobre la comida que los peces deben comer ese día. Por ahora, los peces comerán el mismo alimento todos los días.

Kotlin

fun feedTheFish() {

val day = randomDay()

val food = "pellets"

println("Hoy es $day y los peces comen $food")

}

fun main(args: Array<String>) {

feedTheFish()

}

Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

1. Agrega una función llamada randomDay() en “Hello.kt” para seleccionar un día de la semana al azar desde un arreglo y devolverlo.

Kotlin

fun randomDay(): String {

val week = arrayOf("Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo")

return week[Random().nextInt(week.size)]

}

Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

1. Importa la clase Random al principio del archivo para poder usarla en la función randomDay().

Kotlin

import java.util.\*

Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

1. Ejecuta tu programa y verifica la salida.

Resultado:

Hoy es Martes y los peces comen pellets

### **Paso 2: Utilizando la expresión** when

Vamos a extender esto un poco más. Cambia el código para seleccionar diferentes alimentos para diferentes días utilizando una expresión when. Esta expresión es similar a un switch en otros lenguajes de programación, pero se rompe automáticamente al final de cada rama.

1. Agrega una función llamada fishFood(day: String) que tome un día de la semana como argumento y devuelva el alimento correspondiente para los peces ese día.

Kotlin

fun fishFood(day: String): String {

return when (day) {

"Lunes" -> "hojuelas"

"Miércoles" -> "gusanos rojos"

"Jueves" -> "gránulos"

"Viernes" -> "mosquitos"

"Domingo" -> "plancton"

else -> "nada"

}

}

Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

1. Ejecuta tu programa varias veces para ver diferentes resultados.

Resultado:

Hoy es Jueves y los peces comen gránulos

1. Añade una rama por defecto a la expresión when utilizando else. Para probar, elimina las ramas de “Martes” y “Sábado” para asegurarte de que se tome la rama por defecto en algunas ocasiones.
2. Como ahora tienes una rama por defecto, no necesitas inicializar la variable food antes de asignarle un valor. Puedes declararla con val en lugar de var.
3. ¡Listo! Ahora tu código es más conciso y legible.

Resultado:

Hoy es Martes y los peces comen nada

En resumen, la expresión when es una herramienta poderosa para tomar decisiones basadas en diferentes condiciones y simplificar tu código

4.

## **Valores por Defecto y Funciones Compactas en Kotlin**

### **Paso 1: Creando un Valor por Defecto para un Parámetro**

En Kotlin, puedes pasar argumentos por el nombre del parámetro. También puedes especificar valores por defecto para los parámetros: si un argumento no es proporcionado por el llamador, se utiliza el valor por defecto. Esto evita tener que escribir muchas versiones sobrecargadas del mismo método.

1. Escribe una función llamada swim(speed: String = "fast") que imprima la velocidad de nado de los peces. El parámetro tiene un valor por defecto de “fast”.

Kotlin

fun swim(speed: String = "fast") {

println("nado $speed")

}

Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

1. Llama a la función de tres formas diferentes desde la función main(). Primero, llama a la función usando el valor por defecto. Luego, llama a la función y pasa el parámetro sin nombre, y finalmente, llama a la función nombrando el parámetro.

Resultado:

nado fast

nado slow

nado turtle-like

### **Paso 2: Agregando Parámetros Obligatorios**

Si no se especifica un valor por defecto para un parámetro, el argumento correspondiente siempre debe ser proporcionado.

1. Escribe una función llamada shouldChangeWater(day: String, temperature: Int = 22, dirty: Int = 20): Boolean que tome tres parámetros: day, temperature y dirty. La función devuelve true si el agua debe cambiarse, lo cual ocurre si es domingo, si la temperatura es demasiado alta o si el agua está muy sucia. El día de la semana es obligatorio, pero la temperatura tiene un valor por defecto de 22 y el nivel de suciedad tiene un valor por defecto de 20.

Kotlin

fun shouldChangeWater(day: String, temperature: Int = 22, dirty: Int = 20): Boolean {

return when {

temperature > 30 -> true

dirty > 30 -> true

day == "Domingo" -> true

else -> false

}

}

Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

1. Llama a la función desde feedTheFish() y proporciona el día. El parámetro no tiene un valor por defecto, por lo que debes especificar un argumento. Los otros dos parámetros de shouldChangeWater() tienen valores por defecto, por lo que no es necesario pasar argumentos para ellos.

Resultado:

Hoy es Jueves y los peces comen gránulos

Cambiar agua: false

### **Paso 3: Creando Funciones Compactas**

La expresión que escribiste en el paso anterior condensa mucha lógica en una pequeña cantidad de código. Si quisieras desglosarlo un poco o si las condiciones a verificar fueran más complicadas, podrías usar variables locales con nombres descriptivos. Sin embargo, la forma de hacerlo en Kotlin es mediante funciones compactas.

Las funciones compactas, o funciones de una sola expresión, son un patrón común en Kotlin. Cuando una función devuelve los resultados de una sola expresión, puedes especificar el cuerpo de la función después de un símbolo = y omitir las llaves {} y el return.

En “Hello.kt”, agrega funciones compactas para probar las condiciones.

Kotlin

fun isTooHot(temperature: Int) = temperature > 30

fun isDirty(dirty: Int) = dirty > 30

fun isSunday(day: String) = day == "Domingo"

Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

Resultado:

Hoy es Jueves y los peces comen gránulos

Cambiar agua: false

En resumen, las funciones compactas te permiten escribir código más conciso y legible en Kotlin.

### **Funciones de Transformación y Evaluación Eager vs. Lazy**

Para comprender mejor la evaluación eager y lazy, vamos a utilizar la función map(). Esta función realiza una transformación simple en cada elemento de la secuencia.

1. Utilizaremos la misma lista de decoraciones de acuario que mencionaste anteriormente. Crearemos una transformación con map() que no hace nada y simplemente devuelve el elemento que se le pasó. Además, mostraremos cada vez que se accede a un elemento y asignaremos la secuencia a una variable llamada lazyMap.

Kotlin

val lazyMap = decorations.asSequence().map {

println("acceso: $it")

it

}

Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

Resultado:

acceso: roca

acceso: pagoda

acceso: planta de plástico

acceso: cocodrilo

acceso: maceta

1. Imprimiremos lazyMap, luego mostraremos el primer elemento de lazyMap utilizando first(), y finalmente, convertiremos lazyMap en una lista utilizando toList().

Resultado:

lazy: kotlin.sequences.TransformingSequence@5ba23b66

-----

acceso: roca

first: roca

-----

acceso: roca

acceso: pagoda

acceso: planta de plástico

acceso: cocodrilo

acceso: maceta

all: [roca, pagoda, planta de plástico, cocodrilo, maceta]

1. Ahora crearemos una nueva secuencia utilizando el filtro original antes de aplicar map(). Imprimiremos ese resultado también.

Kotlin

val lazyMap2 = decorations.asSequence().filter { it[0] == 'p' }.map {

println("acceso: $it")

it

}

println("-----")

println("filtrado: ${lazyMap2.toList()}")

Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

Resultado:

-----

acceso: pagoda

acceso: planta de plástico

filtrado: [pagoda, planta de plástico]

1. Otra función de transformación útil para las colecciones en Kotlin es flatten(). Esta función crea una lista a partir de una colección de colecciones, como un arreglo de arreglos o una lista de listas. Vamos a crear una lista de listas y aplicar flatten() para transformar todas las listas en una sola lista.

Kotlin

val mysports = listOf("baloncesto", "pesca", "correr")

val myplayers = listOf("LeBron James", "Ernest Hemingway", "Usain Bolt")

val mycities = listOf("Los Ángeles", "Chicago", "Jamaica")

val mylist = listOf(mysports, myplayers, mycities) // lista de listas

println("-----")

println("Combinada: ${mylist.flatten()}")

Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

Resultado:

-----

Combinada: [baloncesto, pesca, correr, LeBron James, Ernest Hemingway, Usain Bolt, Los Ángeles, Chicago, Jamaica]

En resumen, la evaluación eager crea una nueva lista inmediatamente, mientras que la evaluación lazy espera hasta que se le pida que se evalúe.

5.

## **Comenzando con Filtros en Kotlin**

### **Paso 1: Creando un Filtro**

En Kotlin, los filtros son una forma práctica de obtener parte de una lista basada en alguna condición. Vamos a crear un filtro para seleccionar elementos de una lista de decoraciones de acuario que comiencen con la letra ‘p’.

1. Define una lista de decoraciones de acuario al nivel superior en “Hello.kt”:

Kotlin

val decorations = listOf("roca", "pagoda", "planta de plástico", "cocodrilo", "maceta")

Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

1. Crea una nueva función que imprima solo las decoraciones que comienzan con la letra ‘p’. El código para la condición del filtro está dentro de llaves {}, y se refiere implícitamente a cada elemento mientras el filtro recorre la lista. Si la expresión devuelve true, el elemento se incluye.

Kotlin

fun main() {

println(decorations.filter { it[0] == 'p' })

}

Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

Resultado:

[pagoda, planta de plástico]

### **Paso 2: Comparando Filtros Eager y Lazy**

En Kotlin, los filtros pueden ser tanto eager (inmediatos) como lazy (perezosos), según lo que necesites. Por defecto, los filtros son eager, lo que significa que se crea una nueva lista cada vez que se utiliza el filtro. Sin embargo, puedes hacer que el filtro sea lazy utilizando una secuencia (Sequence), que es una colección que solo puede examinar un elemento a la vez, desde el principio hasta el final.

1. Para hacer el filtro lazy, utiliza una secuencia. Convenientemente, esto es exactamente lo que necesita un filtro lazy.

Kotlin

fun main() {

val decorations = listOf("roca", "pagoda", "planta de plástico", "cocodrilo", "maceta")

// eager, crea una nueva lista

val eager = decorations.filter { it[0] == 'p' }

println("eager: $eager")

// lazy, espera hasta que se le pida evaluar

val filtered = decorations.asSequence().filter { it[0] == 'p' }

println("filtered: $filtered")

}

Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

Resultado:

eager: [pagoda, planta de plástico]

filtered: kotlin.sequences.FilteringSequence@386cc1c4

1. Para forzar la evaluación de la secuencia, conviértela en una lista con .toList().

Kotlin

 // fuerza la evaluación de la lista lazy

val newList = filtered.toList()

println("nueva lista: $newList")

Código generado por IA. Revisar y usar cuidadosamente. [Más información sobre preguntas frecuentes](https://www.bing.com/new#faq).

Resultado:

nueva lista: [pagoda, planta de plástico]



6.

**Lambdas (Funciones Anónimas)**

Imagina que tienes una tarea pequeña que completar, como filtrar una lista o hacer un cálculo sencillo. En lugar de crear una función con nombre completo para este trabajo pequeño, Kotlin te permite usar una expresión lambda. Es como una función rápida y sin nombre que hace el trabajo.

* **Sintaxis:** { parámetros -> código }
  + parámetros: Estas son las entradas que puede recibir la lambda (opcional).
  + ->: Esta flecha separa las entradas del código que se ejecutará.
  + código: Esto es lo que hará la lambda con las entradas.
* **Ejemplo:**
* Kotlin
* 

val sumar = { x: Int, y: Int -> x + y }

val resultado = sumar(5, 3) // resultado será 8

* Usa el código [con precaución](https://gemini.google.com/faq#coding).

## **Creando una Función de Orden Superior**

**Hasta ahora, los ejemplos de lambdas se parecen principalmente a funciones. Sin embargo, el verdadero poder de las lambdas radica en usarlas para crear funciones de orden superior, donde el argumento de una función es otra función.**

**Una función de orden superior es una función que toma otras funciones como parámetros o que devuelve otra función. Puedes pasar una lambda a una función de orden superior que toma una función como argumento. En la tarea anterior, creaste una función de orden superior llamada filter. Pasaste la siguiente expresión lambda a filter como la condición a verificar:**

Kotlin

{ it[0] == 'p' }

**Código generado por IA. Revisar y usar cuidadosamente.** [**Más información sobre preguntas frecuentes**](https://www.bing.com/new#faq)**.**

**De manera similar, map es una función de orden superior, y la lambda que le pasaste fue la transformación a aplicar.**

**Escribiremos una función de orden superior. Aquí tienes un ejemplo básico: una función que toma dos argumentos. El primer argumento es un número entero. El segundo argumento es una función que toma un número entero y devuelve otro número. Pruébalo en el REPL.**

Kotlin

fun updateDirty(dirty: Int, operation: (Int) -> Int): Int {

return operation(dirty)

}

**Código generado por IA. Revisar y usar cuidadosamente.** [**Más información sobre preguntas frecuentes**](https://www.bing.com/new#faq)**.**

**El cuerpo del código llama a la función que se pasó como segundo argumento y le pasa el primer argumento.**

**Para llamar a esta función, pasa un número entero y una función.**

Kotlin

val waterFilter: (Int) -> Int = { dirty -> dirty / 2 }

println(updateDirty(30, waterFilter))

**Código generado por IA. Revisar y usar cuidadosamente.** [**Más información sobre preguntas frecuentes**](https://www.bing.com/new#faq)**.**

**Resultado:**

**15**

**La función que pasas no tiene que ser una lambda; también puede ser una función con nombre regular. Para especificar el argumento como una función regular, utiliza el operador ::. De esta manera, Kotlin sabe que estás pasando la referencia de la función como argumento, no intentando llamar a la función.**

**Intenta pasar una función con nombre regular a updateDirty():**

Kotlin

fun increaseDirty(start: Int) = start + 1

println(updateDirty(15, ::increaseDirty))

**Código generado por IA. Revisar y usar cuidadosamente.** [**Más información sobre preguntas frecuentes**](https://www.bing.com/new#faq)**.**

**Resultado:**

**16**

**Nota: Kotlin prefiere que cualquier parámetro que sea una función se pase como el último parámetro. Al trabajar con funciones de orden superior, Kotlin tiene una sintaxis especial, llamada sintaxis de último parámetro, que te permite hacer el código aún más conciso. En este caso, puedes pasar una lambda para el parámetro de función, pero no necesitas poner la lambda dentro de los paréntesis. Este estilo se considera más idiomático en Kotlin, y lo encontrarás con mayor frecuencia a medida que ganes experiencia con Kotlin.**