Introducción a Kotlin

Herencia

[crayon-6a9d5766e8173345433107/]
Explicación: En este ejemplo, tenemos una clase Animal con un método emitirSonido. La clase Perro hereda de Animal y agrega su propio método ladrar. En la función main, creamos un objeto Perro y llamamos tanto al método heredado emitirSonido como al método propio ladrar.

Polimorfismo

[crayon-6a9d5766e8179399907567/]
Explicación: En este ejemplo, creamos una interfaz Sonido con un método hacerSonido. Las clases Gato y Pajaro implementan esta interfaz y proporcionan su propia implementación del método. En la función main, creamos una lista de animales y llamamos al método hacerSonido para cada uno, lo que demuestra el polimorfismo.

Excepciones

[crayon-6a9d5766e817b339865422/]
Explicación: En este ejemplo, intentamos dividir por cero, lo que genera una excepción ArithmeticException. La excepción se captura en el bloque catch y se imprime un mensaje de error que incluye la descripción de la excepción.

Anotaciones

[crayon-6a9d5766e817d798293651/]
Explicación: En este ejemplo, utilizamos la anotación @Override para indicar que el método miMetodo de la clase MiClase anula un método de la superclase. Esto es útil para

indicar de manera explícita la intención de sobrescribir un método en la herencia.

Variables y tipos de datos

[crayon-6a9d5766e817e985533745/]
Explicación: En este ejemplo, hemos declarado dos variables: nombre y edad. nombre es una variable inmutable (val) que almacena el valor «Juan», mientras que edad es una variable mutable (var) que almacena el valor 25.

Operadores

[crayon-6a9d5766e8180803734810/]
Explicación: En este ejemplo, exploramos operadores en Kotlin. Hemos utilizado el operador + para realizar una suma y el operador == para comparar valores. La variable suma almacena el resultado de la suma 5 + 3, y esIgual almacena true si suma es igual a 8.

Conversión de entero a cadena (Int a String)

[crayon-6a9d5766e8181337345610/]
Explicación: En este ejemplo, convertimos un valor entero (42) en una cadena utilizando la función toString().

Conversión de cadena a entero (String a Int)

[crayon-6a9d5766e8183605827942/]
Explicación: Aquí convertimos una cadena que contiene un número (por ejemplo, «123») en un valor entero utilizando la función toInt().

Conversión de cadena a flotante (String a Float)

[crayon-6a9d5766e8185642324937/]
Explicación: En este caso, convertimos una cadena que representa un número de punto flotante (por ejemplo, «3.14») en un valor flotante utilizando toFloat().

Conversión de flotante a entero (Float a Int)

[crayon-6a9d5766e8186677640227/]
Explicación: En este ejemplo, realizamos una conversión de un valor flotante (por ejemplo, 3.99) a un valor entero. Toma en cuenta que la parte decimal se trunca en este caso.

Conversión de entero a flotante (Int a Float)

[crayon-6a9d5766e8188709137271/]
Explicación: Aquí realizamos la conversión de un valor entero a un valor flotante utilizando la función toFloat().

Conversión de cadena a entero (String a Int) con parseInt

[crayon-6a9d5766e8189307290235/]
Explicación: En este ejemplo, convertimos una cadena que contiene un número (por ejemplo, «42») en un valor entero utilizando Integer.parseInt. La función parseInt es útil para convertir cadenas en valores enteros en Kotlin y manejar posibles excepciones en caso de cadenas no válidas.

Manejo de excepción al convertir una cadena no válida

[crayon-6a9d5766e818b020079686/]
Explicación: En este caso, intentamos convertir una cadena no válida («abc») en un entero utilizando Integer.parseInt. Debido a que la cadena no es un número válido, se lanza una excepción NumberFormatException, que manejamos con un bloque try-catch.

Conversión de cadena con signo (String a Int) con parseInt

[crayon-6a9d5766e818d423591164/]
Explicación: En este ejemplo, convertimos cadenas que incluyen signo («+123» y «-456») en valores enteros utilizando Integer.parseInt. La función maneja los signos correctamente.

Estructuras de Control (if)

[crayon-6a9d5766e818e817699936/]
Explicación: En este ejemplo, utilizamos la estructura de control if para tomar decisiones basadas en condiciones. La variable numero contiene el valor 5. Si número es mayor que 0, se imprime «Número positivo». De lo contrario, se imprime «Número negativo».

Funciones

[crayon-6a9d5766e8190413504113/]
Explicación: En este ejemplo, hemos definido una función llamada saludar que toma un parámetro nombre de tipo String. Dentro de la función, utilizamos println para saludar a la persona cuyo nombre se pasa como argumento. En la función main, llamamos a saludar(«Ana»), lo que imprime «Hola, Ana».

Bucle for simple

[crayon-6a9d5766e8191765836000/]
Explicación: Este bucle for itera a través de los valores del 1 al 5 e imprime un mensaje en cada iteración.

Bucle while

[crayon-6a9d5766e8193124797536/]
Explicación: En este ejemplo, se utiliza un bucle while para imprimir un mensaje en cada iteración mientras se cumple la condición.

Bucle do-while

[crayon-6a9d5766e8194498539640/]
Explicación: El bucle do-while ejecuta el bloque de código al menos una vez y luego verifica la condición. En este caso, se imprime el número del 1 al 5.

Bucle for con lista

[crayon-6a9d5766e8196906075355/]
Explicación: En este ejemplo, se utiliza un bucle for para iterar a través de una lista de nombres y mostrar cada nombre.

Bucle for con índices

[crayon-6a9d5766e8198771649776/]
Explicación: Aquí, se utiliza un bucle for para iterar a través de los índices de un array y mostrar el color en cada índice.

Clases y Objetos

[crayon-6a9d5766e8199434000951/]
Explicación: En este ejemplo, hemos definido una clase llamada Persona con dos propiedades: nombre (inmutable) y edad (mutable). Luego, hemos creado un objeto llamado persona1 de tipo Persona con los valores «Carlos» y 30 para las propiedades. Las clases se utilizan para modelar objetos y su comportamiento.

Colecciones

[crayon-6a9d5766e819b451469987/]
Explicación: En este ejemplo, estamos trabajando con colecciones en Kotlin. Hemos creado una lista llamada listaNombres que contiene los nombres «Ana,» «Juan,» y «Luis.» Además, hemos definido un diccionario llamado diccionario que asocia las claves «clave1» y «clave2» con los valores 10 y 20. Kotlin proporciona colecciones para almacenar datos de manera estructurada.

Null Safety

[crayon-6a9d5766e819c513473832/]
Explicación: En este ejemplo, abordamos el concepto de seguridad en nulos en Kotlin. Hemos declarado una variable llamada nombre de tipo String?, que indica que puede contener un valor nulo. En este caso, hemos asignado null como su valor inicial. Kotlin enfatiza la seguridad en nulos para evitar errores de referencia nula.

Extension functions

[crayon-6a9d5766e819e167881653/]
Explicación: En este ejemplo, presentamos las funciones de extensión en Kotlin. Hemos creado una función de extensión

llamada reverso para la clase String. Esta función invierte el contenido de la cadena. Luego, en la función main, utilizamos esta función de extensión para invertir la cadena «Hola», y el resultado se almacena en la variable reverso.

Tratamiento de excepciones

[crayon-6a9d5766e819f437353462/]
Explicación: En este ejemplo, hemos abordado el manejo de excepciones en Kotlin. Utilizamos un bloque try para ejecutar código que podría generar una excepción. En este caso, intentamos dividir 10 por 0, lo cual genera una excepción aritmética. El bloque catch captura la excepción y permite manejarla. Imprimimos un mensaje de error que incluye la descripción de la excepción.

Entrada y salida

[crayon-6a9d5766e81a1366158111/]
Explicación: En este ejemplo, mostramos cómo interactuar con la entrada y salida de datos en Kotlin. Utilizamos print para solicitar al usuario que ingrese su nombre y readLine para obtener esa entrada. Luego, utilizamos println para saludar al usuario utilizando su nombre.

Iteración y bucles

[crayon-6a9d5766e81a2273997375/]
Explicación: En este ejemplo, demostramos cómo usar un bucle for en Kotlin para realizar una iteración. El bucle imprime «Iteración» seguido del valor de i en cada iteración, desde 1 hasta 5.

Entrada de datos del usuario

[crayon-6a9d5766e81a4747218678/]

Explicación: En este ejemplo, solicitamos al usuario que ingrese su edad utilizando print. Luego, leemos la entrada del usuario con readLine. Dado que readLine devuelve una cadena, la convertimos a un entero utilizando toInt(). Si la entrada es nula, asumimos una edad de 0. Finalmente, mostramos la edad ingresada.

Funciones recursivas

[crayon-6a9d5766e81a5378428727/]
Explicación: En este ejemplo, definimos una función recursiva llamada factorial que calcula el factorial de un número. La función se llama a sí misma de forma recursiva hasta que n llega a 1. El resultado se almacena en la variable resultado y se imprime.

Lectura y escritura de archivos

[crayon-6a9d5766e81a7616102398/]
Explicación: En este ejemplo, importamos la clase File para trabajar con archivos. Creamos un archivo llamado «miarchivo.txt» y escribimos un mensaje en él. Luego, leemos el contenido del archivo y lo imprimimos.

API de Internet (HTTP)

[crayon-6a9d5766e81a8399058725/]
Explicación: En este ejemplo, utilizamos la clase URL para acceder a una API en línea. Abrimos una conexión a la URL, obtenemos la respuesta y la leemos. El resultado es la respuesta de la API que luego imprimimos.

Uso de librerías externas (GSON)

[crayon-6a9d5766e81aa903880769/]
Explicación: En este ejemplo, utilizamos la librería externa

GSON para parsear un JSON. Definimos una clase Persona, creamos un JSON, y luego utilizamos GSON para convertirlo en un objeto Persona, del cual imprimimos sus propiedades.

Programación orientada a objetos (Herencia)

[crayon-6a9d5766e81ab788530555/]
Explicación: En este ejemplo, hemos creado una jerarquía de clases. La clase Animal tiene un método saludar, y la clase Perro hereda de Animal y agrega su propio método ladrar. En la función main, creamos un objeto Perro y llamamos a los métodos.

Programación funcional (Funciones de orden superior)

[crayon-6a9d5766e81ad072236576/]
Explicación: En este ejemplo, utilizamos funciones de orden superior como reduce y map. Reduce se usa para calcular la suma de una lista de números, y map se usa para obtener una lista de los cuadrados de esos números.

Programación concurrente (Hilos)

[crayon-6a9d5766e81ae098966663/]
Explicación: En este ejemplo, hemos utilizado hilos para lograr programación concurrente. Creamos dos hilos que ejecutan iteraciones y los hacemos esperar con join para que se completen antes de continuar.

Expresiones lambda y funciones

anónimas

[crayon-6a9d5766e81b0706591923/]
Explicación: En este ejemplo, hemos definido una expresión lambda que representa una función que toma dos enteros y devuelve su suma. También hemos creado una función anónima con la misma funcionalidad. Luego, llamamos a ambas y mostramos los resultados.

Programación orientada a eventos (Escuchadores)

[crayon-6a9d5766e81b1984857008/]
Explicación: En este ejemplo, hemos creado una interfaz OnClickListener y una clase Boton que tiene un escuchador. Luego, hemos definido una implementación anónima de OnClickListener y asignado esta implementación al botón. Al hacer clic en el botón, se activa el escuchador y se imprime un mensaje.

Uso de anotaciones

[crayon-6a9d5766e81b3843463211/]
Explicación: En este ejemplo, hemos utilizado la anotación @Deprecated para marcar una función como obsoleta y proporcionar una sugerencia de reemplazo. Cuando llamamos a la función obsoleta, se muestra un mensaje de advertencia. Luego, llamamos a la nueva función recomendada.

Uso de extension functions y properties

[crayon-6a9d5766e81b4811349007/]
Explicación: En este ejemplo, hemos agregado extension functions y properties a la clase Rectangulo. La función de

extensión area calcula el área del rectángulo, y la property de extensión perímetro calcula el perímetro, el get() en la propiedad perimetro permite calcular dinámicamente el valor del perímetro cada vez que se accede a esta propiedad sin necesidad de almacenar el resultado previamente. Esto facilita la extensión de clases existentes y la adición de nuevas funcionalidades de manera más flexible. Luego, llamamos a estas extensiones para obtener los resultados.

Tratamiento de errores y excepciones personalizadas

[crayon-6a9d5766e81b6750794162/]
Explicación: En este ejemplo, hemos definido una excepción personalizada DivisionPorCeroException y una función dividir que arroja esta excepción si se intenta dividir por cero. En la función main, intentamos dividir y manejamos la excepción personalizada.

Manejo de listas con funciones de orden superior

[crayon-6a9d5766e81b7202238017/]
Explicación: En este ejemplo, hemos definido una lista de productos y utilizamos funciones de orden superior como sumByDouble para calcular el total de compras y filter para encontrar productos caros. Luego, imprimimos los resultados.

Uso de inicializadores de clase

[crayon-6a9d5766e81b9236041135/]
Explicación: En este ejemplo, hemos creado una clase Persona con propiedades nombre y edad. Utilizamos un inicializador (init) para asignar valores iniciales a estas propiedades. Luego, creamos un objeto Persona e imprimimos sus propiedades.

Uso de companion objects

[crayon-6a9d5766e81bb035172344/]
Explicación: En este ejemplo, hemos creado una clase MiClase con un objeto compañero (companion object). Dentro de este objeto compañero, hemos definido una función mensaje. Luego, hemos llamado a esta función desde la función main.

Uso de lambdas con recorrido de mapas

[crayon-6a9d5766e81bc623025280/]
Explicación: En este ejemplo, hemos definido un mapa de números. Luego, hemos utilizado la función forEach junto con una lambda para recorrer el mapa y mostrar las claves y valores.

Uso de interfaces y herencia múltiple

[crayon-6a9d5766e81be798192198/]
Explicación: En este ejemplo, hemos definido dos interfaces, Nadador y Corredor. Luego, hemos creado una clase Triatleta que implementa ambas interfaces, permitiendo herencia múltiple. La clase Triatleta implementa las funciones nadar y correr, y en la función main, creamos un objeto Triatleta y llamamos a ambas funciones.

Uso de enumeraciones (Enums)

[crayon-6a9d5766e81bf009379843/]
Explicación: En este ejemplo, hemos definido una enumeración DiaSemana que representa los días de la semana. En la función main, hemos asignado el valor MIÉRCOLES a la variable dia y lo hemos impreso.

Uso de funciones de extensión para colecciones

[crayon-6a9d5766e81c1703624930/]
Explicación: En este ejemplo, hemos definido una función de extensión promedio para listas de enteros. Esta función calcula el promedio de los números en la lista. Luego, en la función main, hemos llamado a esta función en una lista de números y mostrado el resultado.

Creación de un objeto con propiedades

[crayon-6a9d5766e81c2340238029/]
Explicación: Hemos creado una clase Coche con propiedades como marca, modelo, y año. Luego, hemos creado un objeto miCoche de esta clase y asignado valores a las propiedades para representar un coche.

Creación de una clase con un constructor

[crayon-6a9d5766e81c4092372488/]
Explicación: En este ejemplo, hemos creado una clase Persona con un constructor que toma nombre y edad como parámetros. Luego, inicializamos las propiedades nombre y edad en el bloque init. Finalmente, creamos un objeto persona con valores proporcionados y mostramos sus propiedades.

Función para verificar un inicio de sesión

[crayon-6a9d5766e81c5136802581/]
Explicación: Hemos definido una función verificarInicioSesion

que toma un objeto Usuario y verifica si las credenciales coinciden con los usuarios registrados en un mapa. Luego, en la función main, hemos creado un objeto usuario y llamado a la función para verificar el inicio de sesión.

Uso de hash tables

[crayon-6a9d5766e81c7641347510/]
Explicación: En este ejemplo, hemos creado una hash table (mutableMapOf) y hemos añadido elementos utilizando la notación de índice. Luego, hemos recorrido la tabla para mostrar las claves y valores.

Uso de listas para gestionar objetos

[crayon-6a9d5766e81c8465184675/]
Explicación: Hemos definido una data class Producto para representar productos con nombre y precio. Luego, hemos creado una lista de productos y la hemos recorrido para mostrar los nombres y precios.

Uso de funciones de extensión para listas

[crayon-6a9d5766e81ca883838417/]
Explicación: Hemos definido una función de extensión promedio para listas de enteros. Esta función calcula el promedio de los números en la lista. Luego, en la función main, hemos llamado a esta función en una lista de números y mostrado el resultado.

Uso de interfaz

[crayon-6a9d5766e81cc539186933/]

Explicación: Hemos definido una interfaz Figura con un método calcularArea. Luego, hemos creado dos clases, Circulo y Cuadrado, que implementan esta interfaz y proporcionan sus propias implementaciones del método. En la función main, hemos creado objetos de ambas clases y calculado sus áreas.

Uso de enumeraciones (juego de piedra, papel o tijera)

[crayon-6a9d5766e81cd466553612/]
Explicación: Hemos definido una enumeración Juego para representar las opciones de Piedra, Papel o Tijera. Luego, hemos creado una función jugar que determina el resultado del juego. En la función main, hemos simulado una partida entre dos jugadores.

Uso de JSON

[crayon-6a9d5766e81cf549556926/]
Explicación: En este ejemplo, hemos definido una data class Persona y la hemos marcado como serializable con la anotación @Serializable. Luego, hemos creado un objeto de tipo Persona y lo hemos convertido a formato JSON utilizando la biblioteca de serialización de Kotlin. El resultado se muestra en la consola.

Uso de listas para gestionar objetos en JSON

[crayon-6a9d5766e81d0054655907/]
Explicación: Hemos definido una data class Producto y la hemos marcado como serializable. Luego, hemos creado una lista de productos y la hemos convertido a formato JSON. El resultado muestra la lista de productos en formato JSON.

Uso de herencia en Kotlin (ejemplo sobre figuras geométricas)

[crayon-6a9d5766e81d2727533825/]
Explicación: Hemos creado una clase base Figura con un método describir, que puede ser sobrescrito. Luego, hemos creado una clase Cuadrado que hereda de Figura y sobrescribe el método. En la función main, hemos creado instancias de ambas clases y llamado al método describir.

Uso de funciones de orden superior (higher-order functions)

[crayon-6a9d5766e81d3157363007/]
Explicación: En este ejemplo, hemos definido una función operar que toma dos números enteros y una función como argumento. Esta función de orden superior realiza una operación con los números proporcionados y la función dada. En la función main, hemos llamado a operar con funciones de suma y resta como argumentos.

Uso de claves (hash tables) para almacenar objetos JSON

[crayon-6a9d5766e81ed898670040/]
Explicación: Hemos definido una data class Producto y la hemos marcado como serializable. Luego, hemos creado dos objetos producto1 y producto2. Hemos utilizado una hash table (mutableMapOf) para almacenar estos objetos en formato JSON con claves. Hemos recorrido la tabla para mostrar las claves y valores en formato JSON.

Uso de funciones closures

[crayon-6a9d5766e81ef856319431/]
Explicación: Hemos definido una función contador que devuelve una función lambda. Esta función lambda actúa como un cierre (closure) y mantiene un contador interno. En la función main, hemos creado una instancia de contador y llamado a la función lambda para incrementar el contador.

Uso de herencia y polimorfismo

[crayon-6a9d5766e81f0563927554/]
Explicación: Hemos creado una clase base Figura con un método calcularArea que puede ser sobrescrito. Luego, hemos creado dos clases, Circulo y Cuadrado, que heredan de Figura y proporcionan sus propias implementaciones del método calcularArea. En la función main, hemos creado instancias de ambas clases y llamado al método calcularArea. Esto demuestra el polimorfismo, donde las instancias de subclases se comportan como instancias de la clase base.

Uso de funciones de extensión en clases

[crayon-6a9d5766e81f2054110084/]
Explicación: Hemos definido una función de extensión reverso para la clase String, que invierte la cadena. En la función main, hemos creado una cadena palabra y llamado a la función de extensión reverso para obtener la cadena al revés.

Uso de colecciones y mapas

[crayon-6a9d5766e81f5785519703/]
Explicación: Hemos creado ejemplos de colecciones en Kotlin, incluyendo una lista de nombres, un conjunto de números y un diccionario (mapa) con claves y valores. Luego, hemos mostrado

el contenido de estas colecciones.

Uso de funciones de extensión y funciones de orden superior

[crayon-6a9d5766e81f7368736574/]
Explicación: Hemos definido una función de extensión filtrar para listas de enteros. Esta función toma una función de orden superior como argumento para determinar qué elementos se deben incluir en la lista resultante. Luego, en la función main, hemos utilizado esta función de extensión para filtrar números pares y números mayores que cinco.

Uso de clases selladas (sealed classes)

[crayon-6a9d5766e81f8097721748/]
Explicación: Hemos definido una clase sellada Resultado que tiene dos subclases: Exito y Error. En la función procesarResultado, utilizamos una expresión when para manejar diferentes tipos de resultados de manera segura. Luego, en la función main, creamos instancias de éxito y error y las procesamos.

Uso de clases abstractas e interfaces

[crayon-6a9d5766e81fa898943889/]
Explicación: Hemos definido una clase abstracta Figura con un método abstracto area() y una interfaz Dibujable con un método dibujar(). Luego, la clase Circulo implementa ambas, proporcionando una implementación concreta para area() y dibujar().

Uso de propiedades computadas

[crayon-6a9d5766e81fd862313591/]
Explicación: Hemos creado una clase Rectangulo con propiedades ancho y alto. La propiedad area se ha definido como una propiedad calculada, utilizando un getter personalizado para calcular el área a partir de ancho y alto.

Uso de operadores sobrecargados

[crayon-6a9d5766e8206929032878/]
Explicación: Hemos creado una clase Punto y sobrecargado el operador + para permitir la suma de dos puntos. Esto nos permite realizar operaciones de suma personalizadas entre objetos de tipo Punto. En esta línea, punto1 + punto2 utiliza el operador +, que se ha sobrecargado mediante la función Punto.plus. La función Punto.plus toma un objeto Punto adicional (otro) y devuelve un nuevo objeto Punto cuyas coordenadas son la suma de las coordenadas de punto1 y punto2. En otras palabras, suma contendrá la suma de las coordenadas x e y de punto1 y punto2.

Uso de generics

[crayon-6a9d5766e8208387691606/]
Explicación: Hemos creado una clase genérica Caja que puede contener cualquier tipo de objeto. Esto nos permite utilizar la misma clase para envolver tanto enteros como cadenas. La T en la declaración class Caja(val contenido: T) se refiere a un tipo genérico. En Kotlin, los tipos genéricos son utilizados para crear clases, funciones o estructuras de datos que pueden trabajar con diferentes tipos de datos de manera flexible. La clase Caja es genérica, lo que significa que puede contener cualquier tipo de dato. La T es simplemente un marcador o nombre de tipo genérico que se utiliza como un comodín para representar un tipo concreto.

En val cajaEntero = Caja(42), la T se infiere automáticamente como Int debido al valor 42 que se pasa como contenido, y en val cajaCadena = Caja(«Hola, Kotlin!»), la T se infiere como String debido al valor «Hola, Kotlin!». Este enfoque genérico permite que la misma clase Caja se utilice con diferentes tipos de datos sin necesidad de crear una clase diferente para cada tipo.

Uso de clases anidadas

[crayon-6a9d5766e8209627746125/]
Explicación: Hemos definido una clase Escuela con una clase anidada Estudiante. Las clases anidadas son útiles para organizar y encapsular conceptos relacionados. Hemos creado una instancia de Estudiante y llamado al método presentarse().

Uso de objetos singleton

[crayon-6a9d5766e820d284174841/]
Explicación: Hemos creado un objeto singleton llamado Registro, que garantiza que solo haya una instancia en todo el programa. Utilizamos este objeto para llevar un historial de eventos, agregar eventos y mostrar el historial.

Uso de operadores infix

[crayon-6a9d5766e820e126716348/]
Explicación: Hemos definido una clase Punto con un operador infix llamado hacia que calcula la distancia entre dos puntos. Luego, en la función main, hemos utilizado este operador de manera más legible.

Uso de expresiones lambda con

receptor

[crayon-6a9d5766e8210064107174/]
Explicación: Hemos creado una función persona que acepta una expresión lambda con receptor y la aplica a una instancia de la clase Persona. Esto permite una construcción más fluida de objetos.

Uso de rangos (ranges)

[crayon-6a9d5766e8211161250506/]
Explicación: Hemos creado rangos numéricos y de caracteres. Los rangos son útiles para verificar si un valor se encuentra dentro de un rango específico.

Uso de colecciones mutable y funciones de modificación

[crayon-6a9d5766e8213127639550/]
Explicación: Hemos creado una lista mutable y utilizado funciones para modificarla. add agrega un elemento, removeAt elimina un elemento por índice y sort ordena la lista.

Uso de mapas (diccionarios) y acceso a elementos

[crayon-6a9d5766e8214376587414/]
Explicación: Hemos creado un diccionario utilizando mapOf y luego accedido a un elemento específico utilizando la clave.

Uso de expresiones when

[crayon-6a9d5766e8216726955690/]
Explicación: Hemos definido una función clasificar que utiliza una expresión when para determinar el tipo de valor y devolver

una descripción.

Uso de corrutinas (coroutines)

[crayon-6a9d5766e8219522132616/]
Explicación: Hemos utilizado el módulo kotlinx.coroutines para crear una corrutina. La corrutina se ejecuta de forma asíncrona y podemos esperar su finalización con job.join().

Uso de expresiones lambda y high-order functions

[crayon-6a9d5766e821b183679728/]
Explicación: Hemos definido una función operar que acepta una lista de números y una función de operación. Usamos una expresión lambda para aplicar la operación a cada elemento de la lista.

Uso de anotaciones (annotations)

[crayon-6a9d5766e821c310979448/]
Explicación: Hemos creado una anotación MiAnotacion y la hemos aplicado a una clase llamada Ejemplo. Luego, hemos utilizado la reflexión para obtener y mostrar el mensaje de la anotación.

Uso de expresiones lambdas con receptores de contexto

[crayon-6a9d5766e821e583346876/]
Explicación: Hemos definido una función persona que acepta una expresión lambda con un receptor de contexto (Persona) y la aplica a una instancia de la clase Persona. Esto permite una construcción más fluida de objetos.

Uso de programación orientada a aspectos

[crayon-6a9d5766e8220873774634/]
Explicación: Hemos utilizado anotaciones (MiAnotacion) para marcar métodos que se ejecutan antes y después de una función principal. Luego, hemos utilizado la reflexión para detectar y ejecutar estos métodos.

Uso de clases selladas (sealed classes)

[crayon-6a9d5766e8221898373798/]
Explicación: Hemos definido una clase sellada (Resultado) con dos subclases (Exito y Error). Esta estructura es útil para representar resultados en un estilo seguro y exhaustivo.

Uso de inicializadores de instancia

[crayon-6a9d5766e8225512662953/]
Explicación: Hemos creado una clase Libro con propiedades titulo y autor. Dentro del inicializador de instancia (init), hemos asignado valores iniciales a estas propiedades.

Uso de enumeraciones con métodos

[crayon-6a9d5766e8226419243926/]
Explicación: Hemos definido una enumeración DiaSemana que representa los días de la semana. También hemos agregado un método esDiaLaboral que indica si un día es laboral o no.

Uso de extension properties

[crayon-6a9d5766e8228930992656/]
Explicación: Hemos creado una propiedad de extensión longitud

para la clase String que calcula la longitud del texto. Esto permite agregar propiedades a clases existentes.

Uso de expresiones lambda y funciones de orden superior con colecciones

[crayon-6a9d5766e8229717379556/]
Explicación: Hemos creado una lista de productos y utilizado expresiones lambda y funciones de orden superior para filtrar los productos caros (precio > 500), aplicar un impuesto (precio * 1.1) y calcular el total.

Uso de expresiones regulares (Regex)

[crayon-6a9d5766e822b951173977/]
Explicación: Hemos utilizado una expresión regular para encontrar números en un texto. La función findAll devuelve todas las coincidencias y las hemos impreso.

Uso de conversión de tipos y seguridad de tipos

[crayon-6a9d5766e822e328552027/]
Explicación: Hemos creado una función imprimirLongitud que demuestra la conversión segura de tipos en Kotlin. Si el argumento es una cadena, calcula su longitud; de lo contrario, muestra un mensaje.

Uso de extension functions con

receptores de contexto

[crayon-6a9d5766e822f806329326/]
Explicación: Hemos creado una extensión de la clase Usuario que agrega la función saludar(). Esto permite que los objetos de tipo Usuario llamen a esta función como si fuera un método de la clase.

Uso de métodos con argumentos con nombre

[crayon-6a9d5766e8231103604178/]
Explicación: Hemos definido una función saludar con argumentos con nombre. Esto permite especificar los argumentos en un orden diferente y proporcionar valores predeterminados.

Uso de propiedades delegadas

[crayon-6a9d5766e8232988462730/]
Explicación: Hemos utilizado propiedades delegadas para observar cambios en la temperatura. Cuando se cambia el valor, se dispara un evento que muestra la temperatura anterior y nueva.

Uso de lambdas con with y apply

[crayon-6a9d5766e8234179014298/]
Explicación: Hemos utilizado las funciones de orden superior with y apply para realizar operaciones en una instancia de la clase Persona. with se usa para múltiples operaciones en la misma instancia, mientras que apply permite realizar operaciones y devolver la instancia modificada.

Uso de la librería estándar

[crayon-6a9d5766e8235906105753/]
Explicación: Hemos utilizado las funciones de la librería estándar de Kotlin, como filter, map, y reduce, para realizar operaciones en una lista de números. En este caso, hemos filtrado los números pares, elevado al cuadrado y sumado.

Uso de la expresión when con múltiples condiciones

[crayon-6a9d5766e8239398338558/]
Explicación: Hemos utilizado la expresión when con múltiples condiciones para determinar el tipo de dato y mostrar un mensaje correspondiente. Si el tipo no se reconoce, se muestra un mensaje predeterminado.

Uso de cláusulas try y catch con recursos (try with resources)

[crayon-6a9d5766e823a332662297/]
Explicación: Hemos utilizado las cláusulas try y catch para manejar excepciones al leer un archivo. Además, hemos utilizado la cláusula finally para asegurarnos de que el recurso se cierre correctamente.

Uso de expresiones de rango (range expressions)

[crayon-6a9d5766e823c622742740/]
Explicación: Hemos utilizado expresiones de rango para iterar sobre números y crear sublistas. En el ejemplo, hemos iterado del 1 al 5 y creado una sublista de nombres.

Uso de object expressions

[crayon-6a9d5766e823d599205004/]
Explicación: Hemos utilizado object expressions para crear una instancia anónima que implementa la interfaz Saludador. Esto nos permite definir una implementación personalizada de la interfaz sin crear una clase separada.

Uso de expresiones sealed y beneficios en jerarquías de clases

[crayon-6a9d5766e823f124814793/]
Explicación: Hemos utilizado la expresión sealed para crear una jerarquía de clases sellada que representa formas geométricas. Esto permite que el compilador verifique que todas las subclases se manejen en la expresión when. Luego, calculamos el área de diferentes formas geométricas.

Uso de anotaciones y reflección

[crayon-6a9d5766e8242253542210/]
Explicación: Hemos creado una anotación personalizada MiAnotacion y aplicado esta anotación a la clase MiClase. Luego, utilizamos la reflección para obtener la anotación y su mensaje asociado.

Uso de corutinas (coroutines) para tareas asíncronas

[crayon-6a9d5766e8244502930768/]
Explicación: Hemos utilizado corutinas para realizar una tarea asincrónica (simulada con delay) sin bloquear el hilo principal. Esto permite que otras tareas se ejecuten mientras la corutina está en espera.

Delegados personalizados

[crayon-6a9d5766e8245662955318/]
Explicación: Hemos creado un delegado personalizado MiDelegado que permite controlar el acceso y modificación de una propiedad en una clase. Esto es útil para implementar lógica personalizada al establecer o acceder a valores.

Proyecciones de tipos genéricos (Type projections)

[crayon-6a9d5766e8247544851614/]
Explicación: Hemos utilizado proyecciones de tipos genéricos (List<*) para permitir que la función imprimirLista imprima listas de diferentes tipos sin conocer el tipo exacto en tiempo de compilación.

Uso de expresiones regulares (Regex)

[crayon-6a9d5766e824a592402120/]
Explicación: Hemos utilizado expresiones regulares (regex) para buscar patrones de correos electrónicos en un texto. Kotlin admite el uso de expresiones regulares de manera efectiva.

Infix functions

[crayon-6a9d5766e824c329558018/]
Explicación: Hemos definido una función infix llamada sumar que permite realizar una suma en notación infija, lo que hace que la llamada sea más legible y natural.

Uso de destructuring declarations

[crayon-6a9d5766e824d084731364/]
Explicación: Hemos utilizado la funcionalidad de declaraciones de destructuración para extraer los componentes de un objeto de datos (Punto) de manera concisa.

Concurrent collections

[crayon-6a9d5766e824f383690034/]
Explicación: Hemos utilizado CopyOnWriteArrayList para crear una colección concurrente que permite la modificación segura mientras se itera sobre ella en un hilo separado.

Programación funcional

[crayon-6a9d5766e8251532805910/]
Explicación: Hemos utilizado funciones de la programación funcional como map, filter, y reduce para transformar, seleccionar y combinar elementos de una lista.

Uso de data classes y copy

[crayon-6a9d5766e8252277506246/]
Explicación: Hemos creado una data class Persona y utilizado la función copy para crear una copia de un objeto con algunos valores modificados.

Uso de rangos (Ranges)

[crayon-6a9d5766e8254186608504/]
Explicación: Hemos creado y utilizado rangos para verificar si un valor está dentro de un rango de números o caracteres.

Patrón delegado por propiedad (Property Delegation)

[crayon-6a9d5766e8257523560790/]
Explicación: Hemos utilizado el patrón de delegado por propiedad para observar cambios en una propiedad y ejecutar una acción cuando cambia su valor.

Uso de extension functions (Funciones de extensión) en librerías externas

[crayon-6a9d5766e8259808194812/]
Explicación: Hemos creado una función de extensión para calcular el promedio de una lista de enteros, lo que demuestra cómo puedes extender clases en bibliotecas externas.

Copiar un archivo en otro: Este código copia el contenido de un archivo de origen en un archivo de destino

[crayon-6a9d5766e825a024942394/]

Lectura de un archivo de texto: Este código lee y muestra el contenido de un archivo de texto

[crayon-6a9d5766e825c267760677/]

Escritura en un archivo de texto: Este código escribe un contenido en un nuevo archivo de texto

[crayon-6a9d5766e825d907849637/]

Copiar un archivo de forma simple: Este código copia el contenido de un archivo de origen en un archivo de destino utilizando una función simple

[crayon-6a9d5766e825f351565853/]

Eliminar un archivo: Este código elimina un archivo si existe

[crayon-6a9d5766e8260448058240/]

Listar archivos en un directorio: Este código lista los archivos en un directorio dado

[crayon-6a9d5766e8264293500600/]

Crear un directorio: Este código crea un nuevo directorio en la ruta especificada

[crayon-6a9d5766e8265913596613/]

Generación de hash (SHA-256)

[crayon-6a9d5766e8267641847919/]
Explicación: En este ejemplo, se genera un hash SHA-256 a partir de un texto utilizando la biblioteca de seguridad de Java.

Cifrado simétrico (AES)

[crayon-6a9d5766e8268699404922/]
Explicación: Este ejemplo muestra cómo cifrar y descifrar un mensaje utilizando el algoritmo AES con una clave secreta.

Cifrado César

[crayon-6a9d5766e826a615911232/]
Explicación: En este ejemplo, se realiza un cifrado César simple que desplaza las letras del alfabeto un número fijo de posiciones hacia adelante.

Firma digital (RSA)

[crayon-6a9d5766e826d849955135/]
Explicación: Este ejemplo muestra cómo firmar y verificar un mensaje utilizando el algoritmo de firma digital RSA.

Cifrado asimétrico (RSA)

[crayon-6a9d5766e826f712130886/]
Explicación: En este ejemplo, se cifra un mensaje utilizando el algoritmo de cifrado asimétrico RSA y luego se descifra con la clave privada correspondiente.

Generación de contraseñas seguras

[crayon-6a9d5766e8270953978532/]
Explicación: Este ejemplo genera una contraseña segura con una longitud específica, utilizando caracteres alfanuméricos y símbolos especiales.

Cifrado simétrico con AES y modo de operación CBC

[crayon-6a9d5766e8272569860982/]
Explicación: En este ejemplo, se utiliza el modo de operación CBC (Cipher Block Chaining) para cifrar y descifrar un mensaje con AES.