🔹 Pregunta 1: Colecciones

Dada la siguiente colección de números:

```
val numeros = mutableListOf(30, 4, 11, 21, 7, 35)
```

Realice las siguientes tareas:

- Cree una lista con todos los elementos **pares.**
- Encuentre el **primer elemento** que sea **menor** a 10.
- Multiplique cada elemento de la lista por 2.
- Sume el **primer** y el **último elemento** de la lista.
- Agregue [4, 4, 4] a la variable números.
- Encuentre el primer elemento que sea divisible por 7
- Sume todos los elementos que sean **pares**.
- Encuentre el primer elemento que sea **impar** y **mayor** a 10.

🔹 Pregunta 2: Ciclos

Considere la siguiente clase y variable:

```
data class Pokemon(
   val nombre: String,
   val nivel: Int,
   val tipo: String
)
```

```
val listaAlumnos = listOf(
    Pokemon("charmander", 5, "fuego"),
    Pokemon("pikachu", 20, "eléctrico"),
    Pokemon("mew", 80, "psíquico"),
    Pokemon("magmar", 70, "fuego")
)
```

Utilice un **ciclo** para recorrer la lista y muestre todos los pokemones que :

- Son de tipo fuego.
- Son de nivel 30 o superior.
- Su nivel sea par.

🔹 Pregunta 3: Condicionales

Dada las variables:

```
val edad = 20
val promedio = 65
val peso = 56
```

Cree un programa que muestre si la persona :

- es mayor de edad o menor de edad
- aprueba la asignatura o no (aprueba con promedio >= 40)
- pesa más de 50 kg
- tiene más de 15 años y pesa más de 50 kg
- si su promedio está entre 60 y 70
- si su edad es múltiplo de 3

🔹 Pregunta 4: Funciones de orden superior

Implemente una función `operacion` que reciba función de suma y resta como parámetro:

```
fun operacion(x: Int, y: Int, f: (Int, Int) -> Int): Int
```

Realice lo siguiente:

- Declare la función `operacion`.
- Declare las funciones `suma` y `multiplicación`.
- Utilice `operacion` para **sumar 2 + 2** y **multiplicar 2 × 2**.

🔹 Pregunta 5: Clases y Herencia

Supongamos que usted debe desarrollar una aplicación de mascotas utilizando **Programación orientada a objeto**, **herencia** y **polimorfismo**.

Para ello realice lo siguiente:

- Cree la clase `Mascota` con los atributos: `nombre: String` y `edad: Int` . Incluya el método `mostrarInfo()` para mostrar la información de esos parámetros.
- Cree dos clases hijas:
    - `Gato`: que incluye `colorPelaje: String`.
    - `Pez`: que incluye `sangreFria: Boolean`.

Ambas clases deben heredar atributos y métodos de `Mascota`. Esto es porque gatos y peces son mascotas.

- Añada los métodos `correr()` y `nadar()` en las clases hijas `Gato` y `Pez` respectivamente. Cada uno debe mostrar un mensaje personalizado. Por ejemplo: gato debe tener un mensaje similar a:

```
println("Miau")
```

Y el pez un mensaje similar a:

```
println("Glue Glue Glue")
```

- Sobreescriba el método `mostrarInfo()` en ambas clases para incluir los nuevos atributos. (colorPelaje y sangreFria)

🔹 Pregunta 6: Control de errores

Supongamos que tenemos el siguiente programa en kotlin:

```
fun main(){
      val nota1 = 50
      val nota2 = 55
      Val nota3 = null

      mostrarpromedio( nota1, nota2, nota3 )
}
```

Usted debe crear la función mostrarPromedio(x, y, z) de tal manera que entregue el promedio de esas notas, pero que sea capaz de funcionar correctamente aún si uno de los parámetros es nulo. Para ello utilice un try catch.

🔹 Pregunta 7: Funciones

Supongamos que tenemos las siguientes variables

```
      val a = 10
      val b = 5
      val c = 2
```

Cree una función que:

- verifique si a es mayor que b
- multiplique a con b
- cuente desde a hasta b
- pregunte si a es impar
- pregunte si a es mayor a 100
- multiplique a x b x c
- diga si a, b y c son todos pares

Profesor: Alvaro Maurelia Vielma
al.maurelia@profesor.duoc.cl