Ejercicios 10-Oct-2025

Funciones básicas y con retorno

1. **Crea una función que imprima “Hola, mundo” al ejecutarse.**

   Pista: Define una función sin parámetros ni retorno, y llámala desde `main()`.

2. **Crea una función que reciba un nombre y muestre un saludo personalizado.**

   Pista: Usa un parámetro `String nombre` y concatenación.

3. **Crea una función que reciba dos números y devuelva su suma.**

   Pista: Usa un `return` con la suma.

4. **Crea una función que reciba tres notas y devuelva su promedio.**

   Pista: Calcula `(a + b + c) / 3` y retorna un `double`.

5. **Crea una función que reciba un número y devuelva `true` si es par y `false` si no.**

   Pista: Usa `% 2 == 0` y devuelve un valor booleano.

6. **Crea una función que reciba un texto y devuelva cuántos caracteres tiene.**

   Pista: Usa la propiedad `.length` de `String`.

7. **Crea una función que calcule el factorial de un número.**

   Pista: Usa un `for` multiplicando desde 1 hasta `n` y retorna el resultado.

8. **Crea una función que reciba dos números y devuelva el mayor.**

   Pista: Usa `if / else` para comparar y retorna el valor mayor.

9. **Crea una función que reciba una lista de precios y devuelva el total.**

   Pista: Suma los elementos con un acumulador.

10. **Crea una función que reciba un texto y devuelva ese texto en mayúsculas.**

    Pista: Usa `.toUpperCase()` y `return`.

Tipos de argumentos

1. **Crea una función con un parámetro opcional que imprima un saludo general o personalizado.**

   Pista: Usa `[String nombre = "invitado"]` como parámetro opcional.

2. **Crea una función con parámetros nombrados que reciba nombre y edad y los imprima.**

   Pista: Usa `{String nombre, int edad}` y llama con `nombre: "Ana", edad: 20`.

3. **Crea una función con un parámetro opcional que muestre un mensaje de error si no se pasa argumento.**

   Pista: Comprueba si el parámetro es `null` con un `if`.

4. **Crea una función que reciba tres parámetros: nombre, edad y país, donde país tenga un valor por defecto.**

   Pista: Usa `{String pais = "España"}`.

5. **Crea una función que reciba una lista y un número y devuelva `true` si la lista contiene ese número.**

   Pista: Usa `.contains()`.

6. **Crea una función que reciba un texto y un número opcional de repeticiones y lo imprima varias veces.**

   Pista: Usa un `for` con el valor del parámetro opcional.

7. **Crea una función que reciba varios números y devuelva su media usando argumentos opcionales.**

   Pista: Usa `[int? b, int? c]` y suma solo los que no sean `null`.

8. **Crea una función que reciba una lista y un índice, y devuelva el elemento correspondiente.**

   Pista: Usa `lista[indice]` y controla con `if` si está fuera de rango.

9. **Crea una función con parámetros nombrados que reciba `nombre`, `edad` y `activo`, y devuelva un mensaje combinando los tres.**

Pista: Usa interpolación de cadenas con `$nombre`.

10. **Crea una función que combine parámetros posicionales y nombrados.**

    Pista: Ejemplo: `void mostrar(String nombre, {int edad = 0})`.

Funciones `async`, `await` y `Future`

1. **Crea una función `async` que devuelva un mensaje después de 2 segundos.**

   Pista: Usa `Future.delayed(Duration(seconds: 2), () => "Mensaje listo");`.

2. **Crea una función `async` que simule descargar datos y luego imprima "Descarga completada".**

   Pista: Usa `await Future.delayed(...)` antes del `print()`.

3. **Crea una función `async` que devuelva el cuadrado de un número después de 1 segundo.**

   Pista: Retorna un `Future<int>` con `await Future.delayed(...)`.

4. **Crea una función que combine `await` y otra función sincrónica.**

   Pista: Llama a una función normal dentro de otra `async` y espera el resultado del `Future`.

5. **Crea una función `async` que recorra una lista y simule procesar cada elemento con una pausa de 1 segundo.**

   Pista: Usa un `for-in` y `await Future.delayed(Duration(seconds: 1))`.

6. **Crea una función que obtenga datos de tres funciones `Future` distintas y las espere con `await`.**

   Pista: Llama a las tres funciones secuencialmente con `await`.

7. **Crea una función `async` que calcule el área de un círculo tras una pausa y devuelva el resultado.**

   Pista: Usa `await Future.delayed()` y retorna `Future<double>`.

8. **Crea una función `async` que simule una validación de login con usuario y contraseña.**

Pista: Usa un `Future.delayed` y un `if` para devolver “Acceso permitido” o “Denegado”.

9. **Crea una función que llame a otra `async` dentro de un bucle `for`, mostrando el progreso paso a paso.**

   Pista: Espera con `await` en cada iteración para que se ejecute en orden.

10. **Crea una función `async` principal (`main() async`) que coordine varias tareas asíncronas y muestre un mensaje al final.**

    Pista: Usa `await` entre llamadas a otras funciones con `Future`.

Manejo de errores con `try` y `catch`

1. **Crea un programa que intente dividir dos números y capture el error si el divisor es cero.**

   Pista: Usa `try` y `catch (e)` alrededor de la operación `a / b`.

2. **Crea una función que reciba dos números y devuelva su cociente, capturando posibles errores.**

   Pista: Usa `try` / `catch` dentro de la función y devuelve `null` si ocurre una excepción.

3. **Crea un programa que convierta un texto a número e intercepte el error si el texto no es válido.**

   Pista: Usa `int.parse()` dentro de un `try` y muestra un mensaje en el `catch`.

4. **Crea una función que lea un número desde teclado y maneje el error si el usuario escribe algo no numérico.**

   Pista: Usa `stdin.readLineSync()` y `try` con `int.parse()`.

5. **Crea un programa que intente acceder a un índice fuera del rango de una lista y capture la excepción.**

   Pista: Usa `try` con `lista[índice]` y muestra el error en el `catch`.

6. **Crea una función que simule leer un archivo y capture cualquier error al “abrirlo”.**

Pista: Usa un `try` con `throw Exception("Archivo no encontrado")`.

7. **Crea una función que pida un número y devuelva su raíz cuadrada, pero capture el error si el número es negativo.**

   Pista: Usa `if (num < 0) throw Exception("Número negativo")` dentro del `try`.

8. **Crea una función `async` que espere un `Future` que falle y capture el error con `try` / `catch`.**

   Pista: Usa `await Future.delayed(...)` seguido de `throw Exception("Error simulado")`.

9. **Crea una función que capture errores distintos según el tipo de excepción.**

   Pista: Usa varios `catch` o un `if (e is FormatException)` dentro del bloque de captura.

10. **Crea un programa que combine `try` / `catch` con condiciones: si hay error, muestra un mensaje distinto según la causa.**

    Pista: Usa un `try` para el código principal y analiza el error dentro del `catch` con `if` / `else`.