Nivel Básico – Fundamentos

1. **Invertir una cadena**
   Escribe una función que reciba un string y devuelva el mismo string pero invertido.
   ***Input*: `"hola"` → *Output*: `"aloh"`**

2. **Número par o impar**
   Crea una función que reciba un número y devuelva `"par"` si es par, o `"impar"` si es impar.

3. **Contar vocales**
   Dado un string, retorna cuántas vocales contiene.
   Input: `"Javascript"` → *Output*: `3`

4. **Sumar elementos de un arreglo**
   Crea una función que tome un array de números y devuelva la suma total.

5. **Eliminar duplicados**
   Escribe una función que reciba un array y retorne uno nuevo sin elementos duplicados.
   Input: `[1, 2, 2, 3, 4, 4]` → *Output*: `[1, 2, 3, 4]`

Nivel Intermedio – Lógica, Estructuras, Métodos

6. **Fibonacci recursivo**
   Devuelve el n-ésimo número de la secuencia de Fibonacci usando recursión.

7. **Validar palíndromo**
   Verifica si una palabra o frase es un palíndromo (se lee igual al revés). Ignora mayúsculas y espacios.
   Input: `"Anita lava la tina"` → *Output*: `true`

8. **Contador de palabras**
   Escribe una función que reciba un texto y devuelva la cantidad de veces que aparece cada palabra.
   Input: `"hola mundo hola"` → *Output*: `{ hola: 2, mundo: 1 }`

9. **Ordenar array de objetos**
   Dado un array de objetos con `nombre` y `edad`, ordénalos de mayor a menor edad.

10. **Simular `Array.prototype.map`**
    Implementa tu propia versión de `map()` que funcione como el método original.

Enunciado: FizzBuzz

Descripción:
Escribe una función en JavaScript llamada `fizzBuzz` que imprima en la consola los números del 1 al 100, pero aplicando las siguientes reglas:

- Si un número es múltiplo de 3, imprime `"Fizz"` en lugar del número.
- Si un número es múltiplo de 5, imprime `"Buzz"` en lugar del número.
- Si un número es múltiplo de ambos, imprime `"FizzBuzz"` en lugar del número.
- En los demás casos, imprime simplemente el número.

Requisitos técnicos:

- Usar una estructura de control (`for` o `while`).
- La función no debe retornar valores, solo imprimir con `console.log`.
- No utilizar librerías externas ni funciones como `eval`.

Bonus:
Haz que la función acepte como argumentos los valores máximos del rango (`n`) y los divisores para `Fizz` y `Buzz`.

Enunciado: Actualización inmutable de objetos

Descripción:
Imagina que estás trabajando en una aplicación que gestiona perfiles de usuarios. Cada perfil se representa como un objeto JavaScript. Para mantener buenas prácticas y evitar efectos colaterales, se te pide **actualizar la información de los usuarios sin modificar el objeto original**.

Debes implementar una función llamada `updateUserProfile` que reciba dos argumentos:

1. `user`: un objeto que representa un perfil de usuario.

2. `updates`: un objeto con las propiedades que deben actualizarse.

La función debe **retornar un nuevo objeto** con los datos actualizados, **sin modificar el objeto original**.

Ejemplo:

```
const user = {
  id: 1,
  name: "Lucía",
  age: 28,
  preferences: {
    theme: "light",
    language: "es"
  }
};

const updates = {
  age: 29,
  preferences: {
    theme: "dark"
  }
};

const updatedUser = updateUserProfile(user, updates);

console.log(updatedUser);
/*
{
  id: 1,
```

```
  name: "Lucía",
  age: 29,
  preferences: {
    theme: "dark",
    language: "es"
  }
}
*/

console.log(user.age); // 28 (no debe haber cambiado)
```

Requisitos técnicos:

- No usar `Object.assign` ni librerías externas como Lodash.
- No modificar directamente el objeto original (`user`).
- Realizar una **copia profunda** solo hasta un nivel (no es necesario soportar estructuras anidadas más allá de 1 nivel).
- Puedes asumir que no hay arrays, solo objetos simples y anidados un nivel.

Bonus:

- Implementa una versión genérica que haga una copia profunda de cualquier objeto anidado (más de un nivel).

Enunciado: flattenMatrix

Escribe una función llamada flattenMatrix que reciba como parámetro una matriz bidimensional (un arreglo de arreglos) y devuelva un nuevo arreglo que contenga todos los elementos de la matriz en un solo nivel, manteniendo el orden original.

Ejemplo:

Supón que tienes la siguiente matriz bidimensional:

const bidimensional = [[1, 1, 3], [2, 3, 4], [1, 2, 3]];

Al llamar a flattenMatrix(bidimensional), la función debe devolver:

[1, 1, 3, 2, 3, 4, 1, 2, 3]

Enunciado: MathChallenge

Escribe una función llamada MathChallenge que reciba un número entero positivo num y devuelva el **mínimo número de monedas** necesarias para sumar exactamente ese valor, usando únicamente monedas de los siguientes valores: **1, 5, 7, 9 y 11**.

Si no es posible obtener el valor exacto con esas monedas, la función debe devolver Infinity.

Ejemplo:

Supón que num = 16.
Puedes obtener 16 usando dos monedas: una de 7 y una de 9 (7 + 9 = 16).
Por lo tanto, la función debe devolver 2.

Ejemplos de uso:

```
console.log(MathChallenge(6));  // 2   (por ejemplo: 1 + 5)
console.log(MathChallenge(16)); // 2   (por ejemplo: 7 + 9)
```

Enunciado: StringChallenge

Escribe una función llamada **StringChallenge que reciba un string** con el siguiente formato:
"XXX.XXX.XXX", donde cada **"X" es un dígito del 1 al 9 (sin ceros)**, y los grupos están separados por puntos.

La función debe devolver "true" si se cumplen todas las siguientes condiciones, o "false" en caso contrario:

1. El **string debe tener exactamente tres grupos de tres dígitos** (del 1 al 9), separados por puntos.
2. La **suma** de los dígitos del **primer grupo** debe ser un número **par**.
3. La **suma** de los dígitos del **segundo grupo** debe ser un número **impar**.
4. En cada grupo, el **último dígito debe ser mayor que los dos anteriores** de ese grupo.

Ejemplos de uso:

```
console.log(StringChallenge("224.315.218")); // "true"
console.log(StringChallenge("11.124.667"));  // "false"
console.log(StringChallenge("114.568.112")); // "true"
```