Javascript challenges

Valores, variables, tipos de datos, operadores básicos

1. Calculadora de BMI (Índice de Masa Corporal)

En este desafío, crearás un programa que calcula el Índice de Masa Corporal (BMI, por sus siglas en inglés) a partir del peso y la altura de una persona. Aquí están los requerimientos:

- Crear dos variables para almacenar el peso (en kilogramos) y la altura (en metros). Estos serán los datos de entrada para tu programa.
- Calcula el BMI. El BMI se calcula con la fórmula peso / altura^2.
- Imprime el BMI. Finalmente, imprime el BMI calculado.
- Interpreta el BMI. Según los estándares de la Organización Mundial de la Salud, un BMI menor a 18.5 se considera bajo peso, entre 18.5 y 24.9 se considera normal, entre 25 y 29.9 se considera sobrepeso, y 30 o más se considera obesidad. Imprime un mensaje correspondiente al rango en el que se encuentra el BMI calculado.

2. Conversor de temperaturas

En este desafío, crearás un programa que pueda convertir grados Celsius a Fahrenheit y viceversa. Aquí están los requerimientos:

- Crea dos variables para almacenar la temperatura en grados Celsius y Fahrenheit:
  - Una variable (por ejemplo, temperaturaCelsius) representará la temperatura en grados Celsius.
  - Otra variable (por ejemplo, temperaturaFahrenheit) representará la temperatura en grados Fahrenheit.
- Calcula y almacena la conversión de la temperatura:
  - Para convertir de Celsius a Fahrenheit, usa la fórmula F = C * 9/5 + 32 y almacena el resultado en temperaturaFahrenheit.
  - Para convertir de Fahrenheit a Celsius, usa la fórmula C = (F - 32) * 5/9 y almacena el resultado en temperaturaCelsius.
- Imprime las temperaturas convertidas:
  - Imprime la temperatura en grados Fahrenheit.
  - Imprime la temperatura en grados Celsius.

3. Generador de Nombres de Usuario

Para este desafío, crearás un programa que generará un nombre de usuario único a partir de un nombre y un apellido. Aquí están los requerimientos:

- Crea dos variables para almacenar el nombre y el apellido. Estos serán los datos de entrada para tu programa.
- Genera el nombre de usuario. Un enfoque común para la generación de nombres de usuario es combinar partes del nombre y el apellido y añadir un número al final. Por ejemplo, podrías tomar las tres primeras letras del nombre, las tres primeras letras del apellido, y un número aleatorio para crear un nombre de usuario.
- Imprime el nombre de usuario. Finalmente, imprime el nombre de usuario generado para mostrar el resultado de tu programa.

4. Juego de Adivinanzas

En este desafío, vas a crear un simple juego de adivinanzas. Aquí están los requerimientos:

- Genera un número aleatorio entre 1 y 10: Este será el número que el jugador debe adivinar.
- Crea una variable para almacenar la suposición del jugador: Para este desafío, puedes asignar un valor fijo a esta variable. Sin embargo, en un programa real, probablemente obtendrías este valor de alguna entrada del usuario.
- Comprueba si la suposición del jugador es correcta:
  - Si el jugador adivina el número, imprime un mensaje de felicitación.
  - Si la suposición es demasiado alta, imprime un mensaje que indique que la suposición es demasiado alta.
  - Si la suposición es demasiado baja, imprime un mensaje que indique que la suposición es demasiado baja.
- Permite al jugador tener un máximo de 3 intentos para adivinar el número. Si no adivinan el número en 3 intentos, informa al jugador que ha perdido y revela el número.

5. Agenda Telefónica

En este desafío, crearás una simple agenda telefónica. Aquí están los requerimientos:

- Crear un objeto para almacenar los contactos de la agenda telefónica. Cada contacto estará representado por un par de propiedades: el nombre de la persona y su número de teléfono.
- Añadir algunos contactos a la agenda. Cada contacto debe ser añadido como una nueva propiedad del objeto de la agenda. El nombre de la persona será la clave, y el número de teléfono será el valor.
- Crea una variable que almacene el nombre de un contacto para buscar.
- Buscar un contacto en la agenda. Si el contacto existe, imprime su número de teléfono. Si no existe, imprime un mensaje indicando que el contacto no se encontró.

6. Simulador de Lanzamiento de Dados

Para este desafío, vas a crear un programa que simula el lanzamiento de un par de dados. Aquí están los requerimientos:

- Crear una variable para cada dado. Cada dado puede tener un valor entre 1 y 6, que puedes generar utilizando una función para obtener números aleatorios.

- Lanzar los dados. Para simular el lanzamiento de los dados, asigna a cada variable un número aleatorio entre 1 y 6.
- Calcula la suma de los valores de los dados.
- Imprime los valores de los dados y su suma. Tu programa debería imprimir un mensaje que muestre los valores de cada dado y la suma de ambos.

7. Validador de Dirección de Correo Electrónico

En este desafío, crearás un programa que valide una dirección de correo electrónico. Aquí están los requerimientos:

- Crear una variable para almacenar la dirección de correo electrónico. Esta será la dirección de correo electrónico que tu programa comprobará.
- Comprobar si la dirección de correo electrónico es válida. Para ser válida, una dirección de correo electrónico debe tener exactamente un símbolo de arroba (@), al menos un punto después del símbolo de arroba, y al menos un carácter antes del símbolo de arroba y después del punto. Este es un chequeo básico y no cubre todas las posibilidades de formatos de correo electrónico válidos, pero servirá para este desafío.
- Imprime un mensaje indicando si la dirección de correo electrónico es válida o no.

8. Calculadora de Propinas

En este desafío, vas a crear un programa que calcula cuánto deberías dejar de propina en un restaurante. Aquí están los requerimientos:

- Crear una variable para almacenar el total de la factura.
- Crear una variable para almacenar el porcentaje de propina que quieres dejar. Esto podría ser un número fijo (por ejemplo, siempre podrías dejar el 15% de propina), o podrías ajustarlo dependiendo de la calidad del servicio.
- Calcula la propina. La propina se calcula como factura * porcentaje / 100.
- Imprime la propina. Finalmente, imprime la propina calculada.

9. Cifrado César

En este desafío, crearás un programa que implemente el cifrado César, una técnica simple de cifrado de sustitución en la que cada letra en el texto original es reemplazada por una letra cierto número de posiciones más adelante en el alfabeto. Aquí están los requerimientos:

- Crear una variable para almacenar el mensaje a cifrar.
- Crear una variable para almacenar el desplazamiento. Este es el número de posiciones que cada letra del mensaje será desplazada en el alfabeto.
- Cifra el mensaje. Para cada letra del mensaje, encuentra la letra que está desplazada en el alfabeto y reemplaza la original por ella. Ten en cuenta que debes tratar de manera especial los espacios y otros caracteres no alfabéticos.
- Imprime el mensaje cifrado.

10. Validador de tarjetas de crédito.

En este desafío, vas a crear un validador de tarjetas de crédito utilizando el algoritmo de Luhn, también conocido como "fórmula de módulo 10". Esta es una fórmula sencilla utilizada para validar varios números de identificación; las tarjetas de crédito y débito son un ejemplo común. Aquí están los requerimientos:

- Crear una función que acepte un número de tarjeta de crédito: Esta función deberá convertir el número a un array de dígitos.
- Implementar el algoritmo de Luhn:
  - Comenzando por el segundo dígito desde la derecha, duplica cada segundo dígito.
  - Si el doble de un dígito resulta ser un número de dos dígitos, suma esos dos dígitos para obtener un número de un solo dígito.
  - Suma todos los dígitos.
  - Si la suma total termina en 0 (es decir, es divisible por 10), entonces el número es válido.
- Probar la función: Prueba la función con varios números de tarjetas de crédito para asegurarte de que funciona correctamente. Recuerda usar solo números de tarjetas de crédito de ejemplo, nunca uses números de tarjetas de crédito reales para estas pruebas.

Strings, String literals, Sentencias if / else

11. Decodificador de mensajes secretos

Tienes una cadena de texto que contiene un mensaje codificado. Cada palabra específica en la cadena se reemplaza por una palabra codificada. Tu trabajo es escribir un programa que recupere el mensaje original.

Por ejemplo, considera la cadena "El gato rojo tiene hambre". Cada palabra codificada tiene su equivalente en una palabra descodificada de la siguiente manera: "gato" debe ser reemplazado por "agente", "rojo" por "secreto", y "hambre" por "misión".

Debes crear una variable para el mensaje codificado y luego cambiar las palabras codificadas por sus correspondientes palabras descodificadas para revelar el mensaje original.

12. Inversor de palabras

Tu tarea es escribir un programa que tome una frase ingresada por el usuario y devuelva la misma frase, pero con todas las palabras invertidas.

Por ejemplo, si la frase es "Hola mundo", el programa debe devolver "aloH odnum".

Nota: El programa debería invertir las palabras individualmente, no la frase entera.

13. Generador de nombres de superhéroe

Tu objetivo es desarrollar un programa que reciba dos entradas del usuario: su nombre y su animal favorito. El programa deberá combinar estas dos entradas para crear y mostrar un nombre de superhéroe único.

Por ejemplo, si el usuario introduce "Juan" como nombre y "águila" como su animal favorito, el programa deberá imprimir "Águila Juan" como el nombre de superhéroe.

Ten en cuenta que el nombre del animal debe ir primero y deberá comenzar con una mayúscula, seguido del nombre del usuario.

14. Verificador de contraseñas

Tu misión es desarrollar un programa que pida al usuario que ingrese una contraseña y verifique si cumple con los siguientes requisitos:

- La contraseña debe tener al menos 8 caracteres de longitud.
- Debe contener al menos una letra mayúscula.
- Debe contener al menos una letra minúscula.
- Debe contener al menos un número.

Si la contraseña ingresada cumple con todos estos requisitos, el programa debe imprimir un mensaje que diga "Contraseña válida". Si no cumple con alguno de los requisitos, el programa debe imprimir un mensaje que indique cuál(es) requisito(s) no se cumplieron.

15. Contador de vocales y consonantes

Tu tarea consiste en desarrollar un programa que solicite al usuario introducir una cadena de texto. Posteriormente, el programa debe contar el número de vocales y consonantes presentes en la cadena de texto proporcionada.

Ten en cuenta lo siguiente:

- El programa no debe ser sensible a las mayúsculas y minúsculas. Es decir, debe contar 'a' y 'A' como la misma vocal, y de manera similar para las consonantes.
- Por simplicidad, puedes asumir que la cadena de texto solo contendrá letras del alfabeto inglés y espacios, no se incluirán números, signos de puntuación u otros caracteres especiales.

16. Verificador de palíndromos

Tu misión es crear un programa que solicite al usuario introducir una cadena de texto y luego compruebe si dicha cadena es un palíndromo. Un palíndromo es una palabra, frase, número o cualquier otra secuencia de caracteres que se lee igual hacia adelante que hacia atrás, ignorando los espacios, la puntuación, y las mayúsculas y minúsculas.

Por ejemplo, "Anita lava la tina" es un palíndromo, porque si se lee de izquierda a derecha o de derecha a izquierda, se obtiene la misma frase (ignorando los espacios y las diferencias entre mayúsculas y minúsculas).

Si la cadena de texto ingresada es un palíndromo, el programa debe imprimir un mensaje que diga "La cadena de texto es un palíndromo". De lo contrario, debe imprimir "La cadena de texto no es un palíndromo".

17. Generador de acrónimos

Tu tarea es escribir un programa que solicite al usuario introducir una frase y, a continuación, genere un acrónimo basado en esas palabras. Un acrónimo es una palabra formada a partir de la inicial o de las primeras letras de varias palabras, y se utiliza como abreviatura.

Por ejemplo, si el usuario introduce "Organización de Naciones Unidas", el programa debería generar el acrónimo "ONU".

Ten en cuenta las siguientes consideraciones para tu programa:

- La frase puede ser de cualquier longitud, pero siempre contendrá al menos una palabra.
- Debes asumir que cada palabra está separada por un espacio.
- Los acrónimos deben estar en mayúsculas.

18. Análisis de mensajes de texto

Tu tarea es escribir un programa que solicite al usuario introducir un mensaje de texto y, a continuación, genere un análisis de dicho mensaje. El análisis debe incluir:

- La cantidad de caracteres en el mensaje, incluyendo espacios y signos de puntuación.
- La cantidad de palabras en el mensaje. Puedes asumir que las palabras están separadas por espacios.
- La cantidad de oraciones en el mensaje. Puedes asumir que las oraciones terminan con un punto, un signo de interrogación o un signo de exclamación.

Por ejemplo, si el usuario introduce "¡Hola, mundo! ¿Cómo estás?", el programa debería informar que el mensaje tiene 29 caracteres, 5 palabras y 2 oraciones.

19. Encriptador simple

Tu tarea es escribir un programa que solicite al usuario introducir un mensaje y luego encripte este mensaje cambiando cada letra por la siguiente en el alfabeto. Por ejemplo, 'a' se convierte en 'b', 'b' se convierte en 'c', etc. Si se encuentra con 'z', debería envolver alrededor y convertirse en 'a'.

Por ejemplo, si el usuario introduce "hola", el programa debería generar el mensaje encriptado "ipmb".

Considera lo siguiente para tu programa:

- El programa debe manejar tanto letras minúsculas como mayúsculas. Por ejemplo, 'A' se convierte en 'B'.
- Cualquier caracter que no sea una letra debe mantenerse igual. Por ejemplo, '!' se mantiene como '!'.

- El encriptado debe ignorar los espacios. Por ejemplo, "hola mundo" se convierte en "ipmb nvoep".

20. Detector de idioma

Tu tarea es escribir un programa que pueda detectar si una palabra ingresada por el usuario es probable que esté en español o en inglés. Este programa funcionará con base en una suposición muy simplificada: si una palabra contiene más vocales tildadas (á, é, í, ó, ú), es probable que sea una palabra en español; si no contiene vocales tildadas, es probable que sea una palabra en inglés.

Para este challenge, asume lo siguiente:

- El usuario introducirá una sola palabra sin espacios.
- El programa debe ser insensible a las mayúsculas y minúsculas. Es decir, debe tratar las letras 'A' y 'a' de la misma manera.

Por ejemplo, si el usuario introduce "Hello", el programa debería deducir que es una palabra en inglés. Si el usuario introduce "película", el programa debería deducir que es una palabra en español.

21."FIZZ BUZZ”

Escribe un programa que muestre por consola (con un print) los números de 1 a 100 (ambos incluidos y con un salto de línea entrecada impresión), sustituyendo los siguientes:

- Múltiplos de 3 por la palabra "fizz".

- Múltiplos de 5 por la palabra "buzz".

- Múltiplos de 3 y de 5 a la vez por la palabra "fizzbuzz".

22. EL "LENGUAJE HACKER"

Escribe un programa que reciba un texto y transforme lenguaje natural a "lenguaje hacker" (conocido realmente como "leet" o "1337"). Este lenguaje se caracteriza por sustituir caracteres alfanuméricos.

- Utiliza esta tabla (https://www.gamehouse.com/blog/leet-speak-cheat-sheet/) con el alfabeto y los números en "leet". (Usa la primera opción de cada transformación. Por ejemplo "4" para la "a")

23. EL PARTIDO DE TENIS

Escribe un programa que muestre cómo transcurre un juego de tenis y quién lo ha ganado. El programa recibirá una secuencia formada por "P1" (Player 1) o "P2" (Player 2), según quien gane cada punto del juego.

- Las puntuaciones de un juego son "Love" (cero), 15, 30, 40, "Deuce" (empate), ventaja.

- Ante la secuencia [P1, P1, P2, P2, P1, P2, P1, P1], el programa mostraría lo siguiente:

15 – Love

30 - Love

30 - 15

30 - 30

40 - 30

Deuce

Ventaja P1

Ha ganado el P1

- Si quieres, puedes controlar errores en la entrada de datos.

- Consulta las reglas del juego si tienes dudas sobre el sistema de puntos.

24. EL GENERADOR DE CONTRASEÑAS

Escribe un programa que sea capaz de generar contraseñas de forma aleatoria. Podrás configurar generar contraseñas con los siguientes parámetros:

- Longitud: Entre 8 y 16.

- Con o sin letras mayúsculas.

- Con o sin números.

- Con o sin símbolos.

(Pudiendo combinar todos estos parámetros entre ellos)

25. PRIMO, FIBONACCI Y PAR

Escribe un programa que, dado un número, compruebe y muestre si es primo, fibonacci y par. Ejemplos:

- Con el número 2, nos dirá: "2 es primo, fibonacci y es par"

- Con el número 7, nos dirá: "7 es primo, no es fibonacci y es impar"

26. PIEDRA, PAPEL, TIJERA, LAGARTO, SPOCK

Crea un programa que calcule quien gana más partidas al piedra, papel, tijera, lagarto, spock.

- El resultado puede ser: "Player 1", "Player 2", "Tie" (empate)

- La función recibe un listado que contiene pares, representando cada jugada.

- El par puede contener combinaciones de " 🗿 " (piedra), " 📄 " (papel), " ✂️ " (tijera), " 🦎 " (lagarto) o " 🖖 " (spock).

- Ejemplo. Entrada: [("🗿","✂️"), ("✂️","🗿"), ("📄","✂️")]. Resultado: "Player 2".

- Debes buscar información sobre cómo se juega con estas 5 posibilidades.

27. EL SOMBRERO SELECCIONADOR

Crea un programa que simule el comportamiento del sombrero selccionador del universo mágico de Harry Potter.

- De ser posible realizará 5 preguntas (como mínimo) a través de la terminal.

- Cada pregunta tendrá 4 respuestas posibles (también a selecciona una a través de terminal).

- En función de las respuestas a las 5 preguntas deberás diseñar un algoritmo que coloque al alumno en una de las 4 casas de Hogwarts: (Gryffindor, Slytherin , Hufflepuff y Ravenclaw)

- Ten en cuenta los rasgos de cada casa para hacer las preguntas y crear el algoritmo seleccionador: Por ejemplo, en Slytherin se premia la ambición y la astucia.

28. EL GENERADOR PSEUDOALEATORIO

Crea un generador de números pseudoaleatorios entre 0 y 100.

- No puedes usar ninguna función "random" (o semejante).

29. HETEROGRAMA, ISOGRAMA Y PANGRAMA

Crea 3 funciones, cada una encargada de detectar si una cadena de texto es un heterograma, un isograma o un pangrama.

- Debes buscar la definición de cada uno de estos términos.

30. VIERNES 13

Crea una función que sea capaz de detectar si existe un viernes 13 en el mes y el año indicados.

- La función recibirá el mes y el año y retornará verdadero o falso.

31. Adivina la palabra

Crea un pequeño juego que consista en adivinar palabras en un número máximo de intentos:
- El juego comienza proponiendo una palabra aleatoria incompleta
    - Por ejemplo "m_ur_d_v", y el número de intentos que le quedan
- El usuario puede introducir únicamente una letra o una palabra (de la misma longitud que la palabra a adivinar)
    - Si escribe una letra y acierta, se muestra esa letra en la palabra. Si falla, se resta uno al número de intentos

- Si escribe una resolución y acierta, finaliza el juego, en caso contrario, se resta uno al número de intentos
- Si el contador de intentos llega a 0, el jugador pierde

- La palabra debe ocultar de forma aleatoria letras, y nunca puede comenzar ocultando más del 60%
- Puedes utilizar las palabras que quieras y el número de intentos que consideres

31. OCTAL Y HEXADECIMAL

Crea una función que reciba un número decimal y lo trasforme a Octal y Hexadecimal.
- No está permitido usar funciones propias del lenguaje de programación que realicen esas operaciones directamente.

32. ANÁLISIS DE TEXTO

Crea un programa que analice texto y obtenga:
- Número total de palabras.
- Longitud media de las palabras.
- Número de oraciones del texto (cada vez que aparecen un punto).
- Encuentre la palabra más larga.

Todo esto utilizando un único bucle.

33. LA TRIFUERZA

¡El nuevo "The Legend of Zelda: Tears of the Kingdom" ya está disponible!

Crea un programa que dibuje una Trifuerza de "Zelda"
formada por asteriscos.
- Debes indicarle el número de filas de los triángulos con un entero positivo (n).
- Cada triángulo calculará su fila mayor utilizando la fórmula 2n-1.

Ejemplo: Trifuerza 2

```
    *
   ***
  *   *
*** ***
```

34. NÚMEROS PRIMOS GEMELOS

Crea un programa que encuentre y muestre todos los pares de números primos gemelos en un rango concreto. El programa recibirá el rango máximo como número entero positivo.

- Un par de números primos se considera gemelo si la diferencia entre ellos es exactamente 2. Por ejemplo (3, 5), (11, 13)
- Ejemplo: Rango 14
  (3, 5), (5, 7), (11, 13)

35. LA ESPIRAL

Crea una función que dibuje una espiral como la del ejemplo.
- Únicamente se indica de forma dinámica el tamaño del lado.
- Símbolos permitidos: ═ ║ ╗ ╔ ╝ ╚

Ejemplo espiral de lado 5 (5 filas y 5 columnas):

36. CUENTA ATRÁS

Crea una función que reciba dos parámetros para crear una cuenta atrás.
- El primero, representa el número en el que comienza la cuenta.
- El segundo, los segundos que tienen que transcurrir entre cada cuenta.
- Sólo se aceptan números enteros positivos.
- El programa finaliza al llegar a cero.
- Debes imprimir cada número de la cuenta atrás.

37. EXPRESIÓN MATEMÁTICA

Crea una función que reciba una expresión matemática (String) y compruebe si es correcta.
Retornará true o false.
- Para que una expresión matemática sea correcta debe poseer un número, una operación y otro número separados por espacios. Tantos números y operaciones como queramos.
- Números positivos, negativos, enteros o decimales.
- Operaciones soportadas: + - * / %

Ejemplos:
"5 + 6 / 7 - 4" -> true
"5 a 6" -> false

38. EL ÁBACO

Crea una función que sea capaz de leer el número representado por el ábaco.

- El ábaco se representa por un array con 7 elementos.

- Cada elemento tendrá 9 "O" (aunque habitualmente tiene 10 para realizar operaciones) para las cuentas y una secuencia de "---" para el alambre.

- El primer elemento del array representa los millones, y el último las unidades.

- El número en cada elemento se representa por las cuentas que están a la izquierda del alambre.

Ejemplo de array y resultado:

["O---OOOOOOOO",

"OOO---OOOOOO",

"---OOOOOOOOO",

"OO---OOOOOOO",

"OOOOOOO---OO",

"OOOOOOOOO---",

"---OOOOOOOOO"]

Resultado: 1.302.790

39. LA COLUMNA DE EXCEL

Crea una función que calcule el número de la columna de una hoja de Excel teniendo en cuenta su nombre.

- Las columnas se designan por letras de la "A" a la "Z" de forma infinita.

- Ejemplos: A = 1, Z = 26, AA = 27, CA = 79.

40. CALCULADORA DE EXPRESIONES RPN

La Notación Polaca Inversa (RPN, por sus siglas en inglés) es una forma de escribir operaciones aritméticas sin utilizar paréntesis. En lugar de escribir la operación entre dos números (por ejemplo, 3 + 4), escribes el número, un espacio, el siguiente número y luego el operador (3 4 +).

Tu tarea es crear una función que evalúe una expresión dada en RPN y devuelva el resultado.

Ejemplo:

3 4 + = 7
5 3 4 + * = 35
5 1 2 + 4 * + 3 - = 14

Requisitos:

La función debe llamarse evaluarRPN y tomar una cadena (la expresión RPN) como argumento.

La cadena siempre será válida y sólo contendrá números enteros, los operadores +, -, * y / y espacios.

Las operaciones deben hacerse de izquierda a derecha.

La función debe devolver el resultado entero de la operación.

Si se encuentra una división entre cero, devuelve Infinity.

Ejemplo de uso:

evaluarRPN("3 4 +"); // devuelve 7
evaluarRPN("5 3 4 + *"); // devuelve 35
evaluarRPN("5 1 2 + 4 * + 3 -"); // devuelve 14

Consejo: Puedes utilizar una pila (stack) para ayudarte a resolver este problema. Por cada número que encuentres, lo apilas. Cuando encuentres un operador, desapilas los números necesarios, realizas la operación, y apilas el resultado.

41. CONVERTIDOR DE FECHAS ROMANAS

El calendario romano original era muy diferente al calendario gregoriano que usamos hoy en día. Sin embargo, para este desafío, nos centraremos solo en representar fechas en formato romano, y no en el funcionamiento real del calendario romano.

Tu tarea es escribir una función que convierta una fecha en formato "DD-MM-YYYY" a su representación romana.

Ejemplo:

01-01-2021 se convertiría en I-I-MMXXI.

Requisitos:

La función debe llamarse fechaRomana y tomar una cadena (la fecha en formato "DD-MM-YYYY") como argumento.

La función debe devolver la fecha en formato romano.

Debes usar los números romanos básicos (I, V, X, L, C, D, M) y considerar que el año nunca excederá 3999 (MMMCMXCIX en romano).

Asume que la entrada siempre es una fecha válida.

Ejemplo de uso:

fechaRomana("01-01-2021"); // devuelve "I-I-MMXXI"
fechaRomana("15-05-1998"); // devuelve "XV-V-MCMXCVIII"

Consejo: Descompone el problema en funciones más pequeñas, por ejemplo, una función que convierta números individuales a su representación romana y luego úsala para construir la fecha completa.

42. CRUCE DE MATRICES

Imagina que tienes dos matrices (arrays bidimensionales) de igual tamaño, A y B. Tu tarea es cruzar estas dos matrices siguiendo un patrón particular:

Por cada celda en la matriz, si la celda en A es un número impar y la celda en B es un número par, coloca un X en la matriz resultante.

Si ambas celdas en A y B son impares o ambas son pares, coloca un O en la matriz resultante.

De lo contrario, coloca un - en la matriz resultante.

Ejemplo:

Dadas las siguientes matrices:

```
A = [[1, 2],
     [3, 4]]

B = [[2, 3],
     [4, 5]]
```

La matriz resultante será:

```
[['X', '-'],
 ['-','O']]
```

Requisitos:

La función debe llamarse cruzarMatrices y tomar dos matrices (arrays bidimensionales) como argumentos.

Ambas matrices tendrán el mismo tamaño.

La función debe devolver una matriz resultante.

Ejemplo de uso:

```
let A = [[1, 2], [3, 4]];
let B = [[2, 3], [4, 5]];
cruzarMatrices(A, B); // Devuelve [['X', '-'], ['-','O']]
```

Consejo: Itera sobre las matrices con dos bucles anidados, uno para las filas y otro para las columnas. Evalúa cada celda según las condiciones dadas y coloca el carácter correspondiente en la matriz resultante.

43. COMPRESIÓN DE CADENAS

Dada una cadena compuesta solo de letras (sin números ni otros caracteres), tu tarea es escribir una función que realice una "compresión básica" de la cadena. La "compresión básica" funciona de la siguiente manera:

Se toma una letra y se cuenta cuántas veces consecutivas aparece.

Si aparece más de una vez, se anexa el número de apariciones después de la letra.

Si solo aparece una vez, solo se escribe la letra.

Tu función debe devolver la cadena comprimida.

Ejemplo:

"aaabcc" se convierte en "a3b1c2"
"abcd" se convierte en "a1b1c1d1" (aunque en este caso la "compresión" no es muy eficiente).

Requisitos:

La función debe llamarse comprimirCadena y tomar una cadena como argumento.

La función debe devolver la cadena comprimida.

Ejemplo de uso:

```
comprimirCadena("aaabcc"); // devuelve "a3b1c2"
comprimirCadena("abcd"); // devuelve "a1b1c1d1"
```

Consejo: Puedes iterar sobre la cadena con un bucle, manteniendo un contador para las apariciones consecutivas de cada letra. Cuando encuentres una letra diferente a la anterior, añade la letra y su contador al resultado y reinicia el contador.

44. ENLAZANDO PALABRAS

La idea de este reto es crear una función que tome una lista de palabras y determine si es posible enlazar todas las palabras de tal manera que la última letra de una palabra sea la primera letra de la siguiente palabra.

Ejemplo:

Con la lista ["apple", "elephant", "tiger", "rat", "tulip"], es posible formar la cadena "apple -> elephant -> tiger -> rat -> tulip". Nota cómo la última letra de cada palabra coincide con la primera letra de la siguiente palabra.

Requisitos:

La función debe llamarse enlazarPalabras y tomar una lista de palabras como argumento.

Si es posible enlazar las palabras, la función debe devolver true; de lo contrario, debe devolver false.

Asume que todas las palabras en la lista son únicas y solo contienen letras en minúsculas.

Ejemplo de uso:

enlazarPalabras(["apple", "elephant", "tiger", "rat", "tulip"]); // devuelve true
enlazarPalabras(["apple", "banana", "cherry"]); // devuelve false

Consejo: Este problema se puede abordar de diferentes maneras. Una forma es intentar encontrar una secuencia de palabras donde cada palabra se enlace con la siguiente usando un enfoque recursivo. Otra manera es tratarlo como un problema de grafos, donde cada palabra es un nodo y hay una arista entre dos nodos si las palabras se pueden enlazar.

45. BUSCANDO SECUENCIAS

La idea detrás de este reto es encontrar todas las secuencias en una lista de números que sumen un valor objetivo. Una secuencia es una serie de números consecutivos en la lista.

Ejemplo:

Dada la lista [1, 2, 3, 4, 5] y un valor objetivo de 9, las secuencias posibles son [2, 3, 4] y [4, 5].

Requisitos:

La función debe llamarse buscarSecuencias y tomar una lista de números y un valor objetivo como argumentos.

La función debe devolver una lista de todas las secuencias posibles que sumen el valor objetivo.

Cada secuencia en la lista resultante debe ser una lista de números.

Si no hay secuencias posibles, devuelve una lista vacía.

Ejemplo de uso:

buscarSecuencias([1, 2, 3, 4, 5], 9); // devuelve [[2, 3, 4], [4, 5]]
buscarSecuencias([1, 2, 3, 4, 5], 1); // devuelve [[1]]
buscarSecuencias([1, 2, 3, 4, 5], 15); // devuelve [[1, 2, 3, 4, 5]]

Consejo: Una forma de abordar este problema es usar dos bucles anidados. El bucle externo marca el inicio de una posible secuencia y el bucle interno prueba diferentes longitudes de secuencias. Suma los números en la secuencia y verifica si coincide con el valor objetivo.