Ejercicios en JavaScript

Objetivos

1. Ampliar los conocimientos sobre el lenguaje de programación JavaScript.
2. Fortalecer las habilidades de análisis, lógica y resolución de problemas aplicando un lenguaje de programación.

Ejercicios

División de números

Dado un número, retornar un array que contenga las dos mitades del número. Si el número es impar, hacer **que el número más a la derecha en el índice sea el número mayor**.

Ejemplos

```
numberSplit(4) -> [2, 2]
numberSplit(10) -> [5, 5]
numberSplit(11) -> [5, 6]
numberSplit(-9) -> [-5, -4]
```

Notas

- Todos los números deben ser enteros.
- Puedes esperar números negativos como parámetros.

Suma de números desde un número singular

Crea una función que tome como argumento un número. Suma todos los números desde 1 hasta el valor que se pase como argumento. Por ejemplo, si el argumento es 4 entonces la función retorna 10 dado que 1 + 2 + 3 + 4 = 10.

Ejemplos

```
addUp(4) -> 10
addUp(13) -> 91
addUp(600) -> 180300
```

Notas

- La función recibe cualquier número positivo entre 1 y 1000.

¿Qué función retorna el número mayor?

Crea una función que reciba como argumentos dos funciones, **f** y **g**, las cuáles no tienen parámetros. Tu función debe llamarlas y retornar un string que indique cuál de las dos funciones retorna el mayor número.

- Si **f** retorna el número mayor, retornar el string **f**.
- Si **g** retorna el número mayor, retornar el string **g**.
- Si ambas funciones retornar el mismo número, retornar el string **ninguna**.

Ejemplos

```
whichIsLarger(() => 5, () => 10) -> "g"
whichIsLarger(() => 25, () => 25) -> "ninguna"
whichIsLarger(() => 505050, () => 5050) -> "f"
```

Notas

Este ejercicio utiliza *higher order functions* (funciones que utilizan otras funciones para realizar su trabajo).

Conversión de Number a Base-2

Crea una función que retorne una representación binaria (base-2) de un número decimal (base-10) convertido en string.

De izquierda a derecha, el valor del bit más a la derecha es 1, a partir de dicho valor cada bit a la izquierda tendrá 2x el valor. El valor de un binario de 8 bit es (256, 128, 64, 32, 16, 8, 4, 2, 1).

Ejemplos

```
binary(1) -> "1"
// 1 * 1 = 1

binary(5) -> "101"
// 1 * 1 + 1 * 4 = 5

binay(10) -> "1010"
// 1 * 2 + 1 * 8 = 10
```

Notas

- Los números siempre serán menores a 1024 (no incluye a 1024).
- El operador **&&** puede ser muy útil.
- El string siempre irá contra la longitud del valor hacia la izquierda en la cual el valor es mayor que el número en **decimal**.
- Si una conversión binaria para 0 es realizada, retornar “0”.

Encontrar el menor y el mayor número

Crea una función que reciba como parámetro un array de números y retorne el número menor y el número mayor, ambos en ese orden.

Ejemplos

```
minMax([1, 2, 3, 4, 5]) -> [1, 5]
minMax([233345, 5]) -> [5, 233345]
minMax([1]) -> [1, 1]
```

Notas

- Todos los arreglos tendrán al menos un elemento válido.

¿Qué se esconde entre la multitud?

¡Una palabra se ha escapado y trata de ocultarse entre una multitud de letras! Ayuda a encontrar la palabra escribiendo una función que cumpla con los siguientes requerimientos:

- La palabra buscada está en **minúsculas**.
- La multitud de letras en la que se oculta está en **mayúsculas**.
- Cabe notar que la palabra está diseminada entre un conjunto de letras aleatorias, pero las letras que la componen **se encuentran en el mismo orden**.

Ejemplos

```
detectWord("UgUNFYGaFYFYGtNUHo") -> "gato"
detectWord("vEEaFGBrFBiRaHgUHbNFYlYeNB") -> "variable"
detectWord("YFjaHUFBvaFBYsFBYLlcGBrYEiFGBMpEtNT") -> "javascript"
```

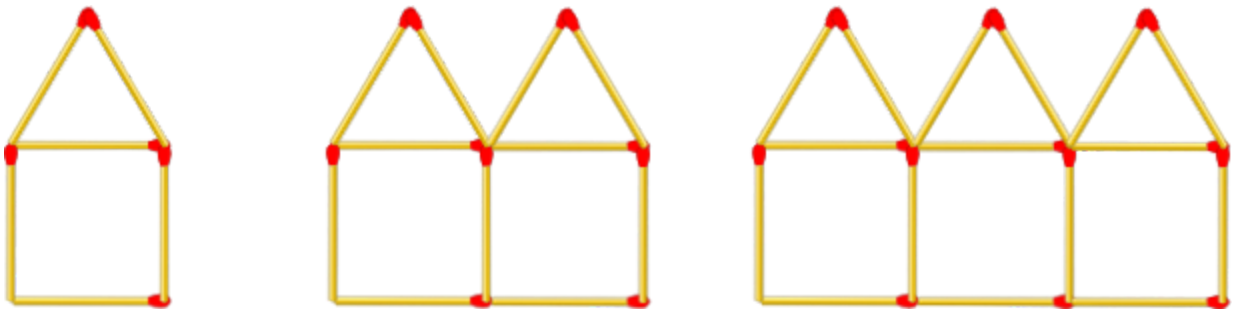
Notas

- N/D

Casas con fósforos

En este ejercicio debes realizar una interpretación matemática de las relaciones algebraicas y geométricas.

Crear una función que reciba como argumento número casas y retorne el número de fósforos requeridos para dichas casas. Puedes tomar como referencia la imagen siguiente.



Ejemplos

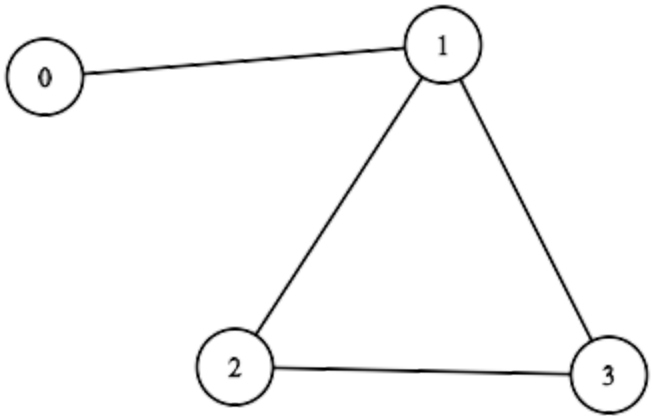
```
matchstickHouses(1) -> 6
matchstickHouses(4) -> 21
matchsitckHouses(87) -> 436
```

Notas

- Si la función recibe 0 como argumento retorna 0.
- El argumento debe ser siempre un valor entero no negativo.
- Puedes pensar en el argumento como el número total de casas que están conectadas.

Encontrando nodos adyacentes

Un grafo es un conjunto de nodos y lados que conectan dichos nodos.



Existen dos tipos de grafos: dirigidos y no dirigidos. En un grafo no dirigido, los lados entre los nodos no tiene una dirección en particular (similar a una calle de ambos sentidos) en cambio, un grafo dirigido, cada lado posee una dirección asociada a él (como una calle de un solo sentido).

Para que dos nodos en un grafo se consideran adyacentes, debe al menos haber un lado que los conecte. En el ejemplo superior, los nodos 0 y 1 son adyacentes, pero los nodos 0 y 2 no lo son.

Podemos codificar grafos utilizando una matriz adyacente. Una matriz adyacente para un grafo de “n” nodos es una matriz de “n * n” dontre la entrada en la fila “i” y la columna “j” es 0 si el nodo “i” y “j” no son adyacentes y 1 si los nodos “i” y “j” son adyacentes.

Para el ejemplo anterior, la matriz de adyacencia es la siguiente:

```
[
  [ 0, 1, 0, 0 ],
  [ 1, 0, 1, 1 ],
  [ 0, 1, 0, 1 ],
  [ 0, 1, 1, 0 ]
]
```

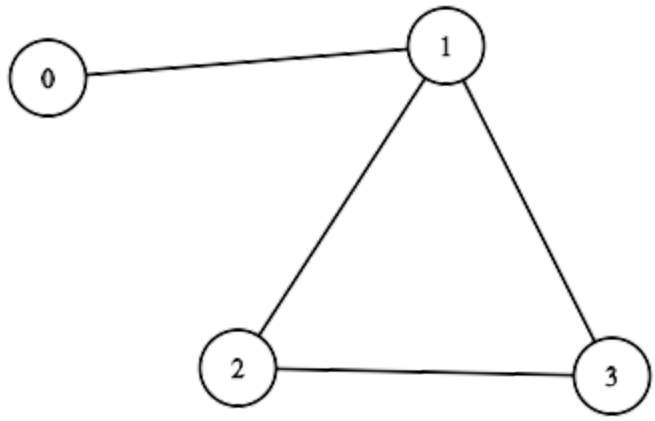
El uno representa que la conexión es verdadera y un cero indica que la conexión es falsa.

La matriz se puede interpretar de la siguiente manera:

- En la primera fila, el nodo 0 no se conecta con el mismo. Pero si se conecta con el nodo 1. No se conecta con el nodo 2 o el nodo 3. La fila se escribe entonces 0, 1, 0, 0.
- En la segunda fila, el nodo 1 conecta con el nodo 0, nodo 2 y nodo 3 pero no se conecta con él mismo. La fila se escribe entonces 1, 0, 1, 1.
- En la tercera fila, el nodo 2 no se conecta con el nodo 0, pero si se conecta con el nodo 1, no se conecta con él mismo pero si lo hace con el nodo 3. La fila se escribe entonces 0, 1, 0, 1.
- En la cuarta fila, el nodo 3 no se conecta con el nodo 0, pero si se conecta con el nodo 1 y el nodo 2, no se conecta con el mismo. La fila se escribe entonces 0, 1, 1, 0.

Escribe una función que determine si dos nodos son adyacentes o no lo son dado que recibes una matriz de adyacencia y dos nodos.

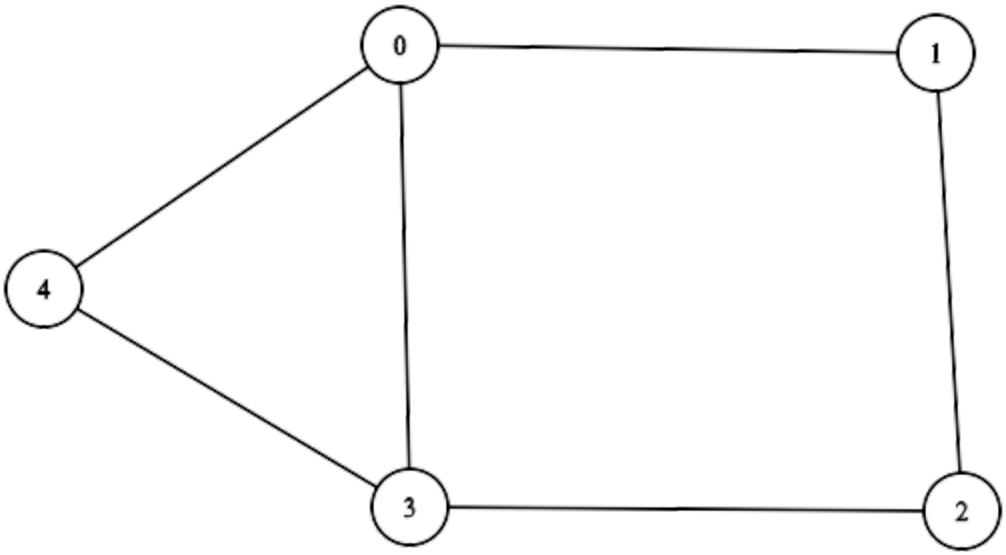
Ejemplos



Matriz de adyacencia:

```
[
  [ 0, 1, 0, 0 ],
  [ 1, 0, 1, 1 ],
  [ 0, 1, 0, 1 ],
  [ 0, 1, 1, 0 ]
]
```

- Nodos `0, 1` deben retornar `true`
- Nodos `0, 2` deben retornar `false`




```
[
  [ 0, 1, 0, 1, 1 ],
  [ 1, 0, 1, 0, 0 ],
  [ 0, 1, 0, 1, 0 ],
  [ 1, 0, 1, 0, 1 ],
  [ 1, 0, 0, 1, 0 ]
]
```

- Nodos `0, 3` deben retornar `true`
- Nodos `1, 4` deben retornar `false`

Notas

- Los grafos pueden tener un límite entre 0 y 25,000 nodos

Suma de arreglos

Dado tres arreglos de enteros retornar la suma de los enteros que son comunes en los tres arreglos.

Ejemplos

```
sumCommon([1, 2, 3], [5, 3, 2], [7, 3, 2]) → 5
// 2 & 3 son comunes en los tres arreglos.

sumCommon([1, 2, 2, 3], [5, 3, 2, 2], [7, 3, 2, 2]) → 7
// 2, 2 & 3 son comunes en los tres arreglos.

sumCommon([1], [1], [2]) → 0
```

Notas

- N/A

Encuentra el n-ésimo número tetraédrico

Un tetraedro es una pirámide con una base triangular y tres lados. Un número tetraédrico es el número de elementos en un tetraedro.

Crea una función que reciba un entero n y retorne el n-ésimo número tetraédrico.

Ejemplos

```
tetra(2) → 4

tetra(5) → 35

tetra(6) → 56
```

Notas

- Existe un fórmula para calcular el n-ésimo número tetraédrico

Ordenar arreglo de números

En este problema se te presenta un arreglo similar al siguiente:

```
[[3], 4, [2], [5], 1, 6]
```

Los elementos de dicho arreglo pueden ser un número entero o un arreglo que contiene solamente un número entero. A simple vista podemos observar que este arreglo se puede ordenar fácilmente por el “contenido de los elementos”de la siguiente manera:

```
[1, [2], [3], 4, [5], 6]
```

Crear una función que, dado un arreglo similar al anterior, ordene dicho arreglo de acuerdo al “contenido de los elementos”.

Ejemplos

```
sortIt([4, 1, 3]) → [1, 3, 4]

sortIt([[4], [1], [3]]) → [[1], [3], [4]]

sortIt([4, [1], 3]) → [[1], 3, 4]

sortIt([[4], 1, [3]]) → [1, [3], [4]]

sortIt([[3], 4, [2], [5], 1, 6]) → [1, [2], [3], 4, [5], 6]
```

Notas

- Los elementos del arreglo solamente serán números enteros o arreglos con un único número entero.

Aplanando un arreglo

John escribe una función para aplanar (flattening) un arreglo de subarreglos en un solo arreglo. En otras palabras, John quiere transformar esto `[[1, 2], [3, 4]]` en esto `[1, 2, 3, 4]`.

Para dicho ejercicio suponemos que John desconoce la función `.flat()` de JavaScript.

Este es el código de John

```
function flatten(arr) {
  arr2 = [];
  for (let i = 0; i < arr.length; i++) {
    arr2.concat(arr[i]);
  }
  return arr2;
}
```

Pero, ¡parece que el código de John no funciona!. Ayuda a John a corregir su código para que aplane de manera correcta el arreglo.

Ejemplos

```
flatten([[1, 2], [3, 4]]) → []
// Resultado esperado: [1, 2, 3, 4]

flatten([["a", "b"], ["c", "d"]]) → []
// Resultado esperado: ["a", "b", "c", "d"]

flatten([[true, false], [false, false]]) → []
// Resultado esperado: [true, false, false, false]
```

Notas

- No se puede utilizar la función .flat()

Morfismo numérico

Un número entero *n* que elevado a la potencia de otro número *k* “finalice” con el mismo número *n*, es un número automórfico.

```
5² = 25
// Es automórfico porque "25" finaliza en "5"

5³  = 125
// Es automórfico porque "125" finaliza en "5"

76⁴ = 33362176
// Es automórfico porque "33362176" finaliza en "76"
```

Un número puede tener varias potencias que lo puede convertir en automórfico (puedes ver el número 5 en el ejemplo anterior). En este ejercicio, debes verificar si dado un número, dicho número es automórfico para cada potencia del 2 al 10.

Dado un número entero positivo *n*, implementa una función que retorne:

- “Polimórfico” si el número es automórfico para cada potencia del 2 al 10.
- “Quadrimórfico” si el número es automórfico para cuatro potencias del 2 al 10.
- “Bimórfico” si el número es automórfico para dos potencias del 2 al 10.
- “Enamórfico” si el número es automórfico para 1 potencia del 2 al 10.
- “Amórfico” si el número no es automórfico para ninguna potencia del 2 al 10.

Ejemplos

```
powerMorphic(5) → "Polimórfico"
// Del 2 al 10, cada potencia de 5 finaliza en 5

powerMorphic(21) → "Enamórfico"
// 21⁶ = 85766121

powerMorphic(7) → "Dimórfico"
// 7⁵ = 716807
// 7⁹ = 40353607

powerMorphic(4) → "Quadrimórfico"
// 4³ = 64
// 4⁵ = 1024
// 4⁷ = 16384
// 4⁹ = 262144
```

```
powerMorphic(10) → "Amórfico"
// No hay potencias que lo hagan anamórfico
```

Notas

- Puedes realizar un ciclo completo para verificar si el número es anamórfico para cada potencia o puedes tratar de encontrar la discriminante que permite que disminuyan la lógica de tu código.
- Debes utilizar BigInt para evitar errores de aproximación.