Para evitar errores en la ejecución y en la corrección, cada uno de los ejercicios deben entregarse en su carpeta correspondiente. En esta carpeta, llamada **ejercicioX** (siendo X el número de ejercicio), se guardarán los archivos **main.js** , **index.html** (opcional) y **styles.css** (opcional) correspondientes a la resolución de este ejercicio. Los archivos html y css no son evaluables ni obligatorios en todos los ejercicios, pero sí son necesarios para la correcta ejecución de algunos de ellos.

Todas las carpetas de cada uno de los ejercicios deben entregarse mediante un archivo comprimido (zip, tar, rar…) que se debe subir a la plataforma. Este archivo debe seguir la siguiente nomenclatura:

1TDAW_DWEC_Apellido1_Apellido2_Nombre_ENTREGA_1.zip

1. Crea un script que pida al usuario un numero entero positivo N mayor a 0. Hay que controlar que el número introducido sea correcto. Si no es así se volverá a pedir.
   A continuación debe realizar lo siguiente:
   - Calcular los divisores del número N y mostrarlos. Un número *y* es divisor de otro *x* si el resultado de efectuar la operación de resto es igual a 0:

     x%y = 0 --> y es divisor de x

   - Calcular la suma de los cuadrados de los divisores obtenidos en el paso anterior y mostrarla.
   - Indicar si esa suma es un cuadrado o no con una frase por pantalla. Se dice que un número es un cuadrado si es el resultado de la multiplicación de un número por sí mismo.

2. Crear un programa JavaScript donde se introduzca un número de kilómetros que corre un Runner al día. Según los kilómetros recorridos a la semana se clasifica a los Runners en las siguientes categorías usando los siguientes intervalos. Suponemos que un Runner corre los 7 días de la semana los mismos kilómetros al día.

   INTERVALO CATEGORIA
   0<KILOMETROS<=10 Corredor novato
   10<KILOMETROS<=30 Corredor iniciado
   30<KILOMETROS<=40 Corredor experto
   KILOMETROS>40 Corredor nivel Élite

Se debe mostrar mediante document.write la categoría del Runner. Si el usuario escribe un número negativo se debe mostrar un mensaje de error.

3. El presupuesto de una obra se distribuye en los siguientes conceptos:
   - 50% de materiales.
   - 20% mano de obras.
   - 30% en licencias de obra.

   Crear un programa JavaScript que pida mediante un prompt el presupuesto total de la obra y que muestre mediante document.write una lista ordenada del coste de cada concepto. Si el usuario introduce un presupuesto negativo mostrar un mensaje de error dentro de un h1.

4. Define un array con los siguientes colores: red, yellow, green, white, blue, brown, pink y black. A continuación, crea un generador aleatorio de banderas:
   - Se pide el número de franjas que va a tener la bandera (entre 1 y 5). Se debe comprobar que el número introducido cumple las características pedidas.
   - El programa obtiene de forma aleatoria 5 colores del array. Para obtener un valor aleatorio se puede utilizar la función de Math random, junto con la función floor. Se utilizan de la siguiente manera:

```
nombreArray[Math.floor(Math.random() * nombreArray.length)]
```

   - Usando document.write, crea una tabla de una fila y tantas columnas como colores tenga la bandera generada. Usa el atributo style para rellenar el fondo de cada celda del color adecuado.
     a. En el paso 2 se pueden repetir colores en la bandera.
     b. En el paso 2 NO se pueden repetir colores en la bandera.
     c. En el paso 2 se pueden repetir colores mientras no sean consecutivos (es decir, no puede haber dos franjas juntas con el mismo color).

5. Usando el array de colores del ejercicio anterior, crea un script que solicite 8 palabras al usuario y las almacene en otro array. Ordena ese array (el del usuario) de forma que, si aparecen colores del array de colores, estos queden al principio del array y el resto de palabras al final. Muéstralo por consola.

   Ejemplo:

   Array de palabras del usuario:

   `casa blue green orden brown bombilla bici pink`

Array resultante:

```
blue green brown pink casa orden bombilla bici
```

6. Escriba un script que, dados dos arrays cualesquiera, devuelva un nuevo array con los elementos que solo aparecen una vez en total (ya sea en el primer o en el segundo array). El orden debe ser: primero los que están en el primer array y luego los que están en el segundo.

   Ejemplos:

   a. [1, 2, 3, 3] y [3, 2, 1, 4, 5]) ==> [4, 5]
   b. ["Ray", "Jose", "Dani"] y ["Dani", "Jose", "Ivan"]) ==> ["Ray","Ivan"]
   c. [77, "ciao"] y [78, 42, "ciao"]) ==> [77, 78, 42]
   d. [1, 2, 3, 3] y [3, 2, 1, 4, 5, 4]) ==> [4,5]

7. Realiza un script que pida 10 números por teclado y que los almacene en un array. A continuación, muestra el contenido de ese array junto al índice de cada número (mira el dibujo).
   Seguidamente el script pedirá dos posiciones a las que llamaremos inicial y final. Debes comprobar que inicial es menor que final y que ambos números están entre 0 y 9. Si no es así, vuelve a pedirlos.
   A continuación, coloca el número de la posición inicial en la posición final, desplazando el resto de números en el array para que no se pierda ninguno.

   Al final se debe mostrar el array resultante.

   Por ejemplo, para inicial = 3 y final = 7:

Índice	0	1	2	3	4	5	6	7	8	9
Valor	20	5	7	4	32	9	2	14	11	6

Array inicial

Índice	0	1	2	3	4	5	6	7	8	9
Valor	6	20	5	7	32	9	2	4	14	11

Array final

8. Un restaurante nos ha encargado una aplicación para colocar a los clientes en sus mesas. En una mesa se pueden sentar de 0 (mesa vacía) a 4 comensales (mesa llena).
El funcionamiento es el siguiente:
Cuando llega un cliente se le pregunta cuántos son. Como el programa no está preparado para colocar a grupos mayores a 4, si un cliente solicita una mesa con más comensales (por ejemplo, 6), el programa dará el mensaje "Lo siento, no admitimos grupos de 6, haga grupos de 4 personas como máximo e intente de nuevo" y volverá a preguntar.

   Para cada grupo nuevo que llega, se busca siempre la primera mesa libre (con 0 personas). Si no quedan mesas libres, se busca una donde haya hueco para todo el grupo (por ejemplo, si el grupo es de dos personas, se podrá colocar en mesas donde haya una o dos personas).

   Cada vez que se sientan nuevos clientes se debe mostrar el estado de las mesas. Los grupos no se pueden romper, aunque haya huecos sueltos suficientes.
   A tener en cuenta:
   - El programa comienza pidiendo el número de mesas que tiene el restaurante.
   - Inicialmente, las mesas se cargan con valores aleatorios entre 0 y 4 y mostrará por pantalla como quedan las mesas inicialmente.
   - El programa seguirá pidiendo comensales hasta que se introduzca un valor negativo.

   Ejemplo de ejecución:

```
//El usuario ha metido un valor de 10

Estado de las mesas: 3 2 0 2 4 1 0 2 1 1

El usuario pide una mesa para 2.
Por favor, Siéntese en la mesa 3

Estado de las mesas: 3 2 2 2 4 1 0 2 1 1

El usuario pide una mesa para 4
Por favor, Siéntese en la mesa 7

Estado de las mesas: 3 2 2 2 4 1 4 2 1 1
```

```
El usuario pide una mesa para 3
Por favor, Siéntese en la mesa 6

Estado de las mesas: 3 2 2 2 4 4 4 2 1 1

El usuario pide una mesa para 4
Lo siento, no queda sitio en el restaurante.

Estado de las mesas: 3 2 2 2 4 1 4 2 1 1
```