Ejercicios Adicionales de Arrays

1. Escribe la función **IntercambiaParImpar**. A esta función le pasaremos un array de enteros por parámetro y en ese mismo array intercambiará los elementos de las posiciones pares por los elementos de las posiciones impares: el elemento de la posición 0 se intercambiará con el de la posición 1, el de la posición 2 con el de la posición 3, etc. El array tendrá que tener un número par de elementos (2, 4, 6, etc.). De no ser así, la función mostrará un mensaje de error y dejará el array como estaba.
   Ej.: Si le pasamos [1, 2, 3, 4, 5, 6], el array quedará: [2, 1, 4, 3, 6, 5]
   Si le pasamos [1, 2, 1, 2, 1, 2], el array quedará: [2, 1, 2, 1, 2, 1]
   Si le pasamos [7, 1, 4, 6, 9, 5], el array quedará: [1, 7, 6, 4, 5, 9]

2. Escribe la función **CompruebaTarjeta**. A esta función le pasaremos un array de enteros con los 16 números de una tarjeta de crédito. La función nos devolverá un *bool* que será *true* si el número de tarjeta es correcto y *false* si no lo es. Para averiguar si un número de tarjeta de crédito es correcto utilizaremos el siguiente algoritmo:
   1. Multiplicar por dos los valores que estén en las posiciones pares del array (0, 2, …).
   2. Si algún valor es mayor que 10, restarle 9 a ese valor.
   3. Sumar todos los valores. Si el resultado es múltiplo de 10, la tarjeta es buena.

   Ejemplos de tarjetas válidas: 5565 8666 4522 7307, 4929 9968 6150 1756

   Nota: No os carguéis el array inicial. Si el array no tiene 16 elementos, devolver *false* directamente.

3. Escribe la función **TailArray**. A esta función le pasaremos un array de enteros y un número *n*. La función nos devolverá otro array de enteros con los *n* últimos elementos del array que le pasamos. O sea, si *n = 5*, nos devolverá un array con las cinco últimas posiciones del primero. Si *n* es mayor que el número de elementos del array o *n* es menor que 0, la función mostrará un mensaje de error y devolverá el array original.
   Ej.: Si le pasamos [1, 2, 3, 4, 5, 6] y n=3, nos devolverá: [4, 5, 6]

   Nota: El array que le pasamos por parámetro debe quedar exactamente igual que al principio; no os lo carguéis.

4. Escribe la función **Detecta5** a la que le pasamos un array de enteros y nos dice si hay, al menos una vez, cinco números iguales consecutivos (nos devuelve un *boolean*).

5. Escribe la función **CalculaNPrimos**. A la función le pasaremos un entero que será el máximo número primo que queremos calcular. Por ejemplo, si le pasamos el 10 nos dirá qué números son primos entre 1 y 10.
   El resultado lo devolverá en un array de booleanos, de tal forma que, si un número x es primo, en la posición x pondrá *true* y si no es primo pondrá *false*.

   Ej.: CalculaNPrimos(10) nos devolverá un array de 11 elementos (porque la primera posición es 0 y necesitamos 11 para que exista la posición 10).

El array será: `[ False False True True False True False True False False False ]`
Las dos primeras posiciones son *false* porque el 0 y el 1 no son primos y luego estarán a *true* las posiciones 2, 3, 5 y 7.

6. Escribe la función **EscribeArrayNotas** que te escribe por pantalla un array de *double* que contiene notas del alumnado, con las siguientes características:
    - Si la nota está entre 5 y 10, se escribirá en verde.
    - Si la nota está entre 0 y 5 (sin incluir el cinco), se escribirá en rojo.
    - Si la nota no está entre 0 y 10, en lugar de la nota se escribirá "ERROR" en amarillo.
    - El resto de cosas (los corchetes, las comas) se escribirán en blanco.

    Para cambiar de color en la consola, antes de escribir con Console.Write tenéis que usar la función "Console.ForegroundColor = ConsoleColor." y te sale una lista para elegir el color que quieres usar.

7. Escribe la función **TachaElementosNoComunes** a la que le pasamos dos arrays. La función sustituirá los elementos de un array que no aparezcan en el otro por ceros (y viceversa).
    Ej: [1, 3, 5, 7] y [1, 2, 3, 4] => [1, 3, 0, 0] y [1, 0, 3, 0]

8. Escribe la función **EliminaRepetidos** a la que le pasas un array y te devuelve otro array en el que se han eliminado los elementos que estén repetidos.

    Ej: [1, 5, 9, 2, 3, 4, 1, 1, 2] => [1, 5, 9, 2, 3, 4]

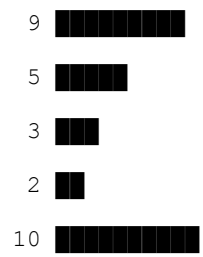
9. Escribe dos funciones: **RedondeaArray** y **DecimalesArray**. A la función **RedondeaArray** le pasaremos un array de *double* y nos devolverá otro array de *double* en el que a los números le quitamos la parte decimal.
    Ej.: `[1.56, 2.75, 3, 9.9] => [1, 2, 3, 9]`
    A la función **DecimalesArray** le pasaremos un array de *double* y nos devolverá otro array de *double* en el que aparecerán solamente las partes decimales de los números.
    Ej.: `[1.56, 2.75, 3, 9.9] => [0.56, 0.75, 0, 0.9]`

10. Escribe la función **DesordenaArray**. La función recibirá un array de enteros por parámetro, que al término de la función deberá quedar desordenado.
    Para conseguir esto, iremos cogiendo los elementos de un array de manera aleatoria y los iremos poniendo en un array auxiliar.

11. Escribe la función **MinMaxArray** a la que le pasaremos un array de enteros y nos devolverá otro array de enteros diferente en el que aparecerán los números comprendidos entre el menos y el mayor del array que le hemos pasado.
    Con un ejemplo se entiende mejor:
    - Le pasamos `[7, 3, 9, 4, 4, 6]`
    - El mínimo es 3, el máximo 9.

    Nos devolverá `[3, 4, 5, 6, 7, 8, 9]`

12. Escribe la función **GraficaBarras** a la que le pasamos un *array* de enteros y nos muestra una gráfica de barras por pantalla. Ejemplo: [9, 5, 3, 2, 10]



- Antes de pintar la gráfica, habrá que comprobar que todos los valores estén entre 0 y 10 (supongamos que representan vuestras notas). Si hay valores incorrectos, no se pinta la gráfica y se muestra un mensaje de error.
- Usar el caracter ASCII 219 (█) para las barritas
- Dejar una línea en blanco entre barra y barra para que quede bonito.
- OPCIONAL: que los aprobados aparezcan en verde y los suspensos en rojo.

13. Escribe la función **MejorRacha** a la que le pasamos un *array* que contiene resultados de fútbol y nos devuelve un entero correspondiente a la mayor racha de victorias seguidas. El *array* contendrá parejas de números (o sea, su tamaño será par) de los cuales el primero serán los goles que ha marcado nuestro equipo y el segundo los goles que ha recibido (p.ej.: [1, 0, 2, 2, 3, 5] se correspondería con una victoria por 1-0, un empate 2-2 y una derrota por 3-5). La función contará cuantas victorias consecutivas hay y nos devolverá la mayor racha.

    Ej.: [2, 1, 1, 0, 1, 1, 0, 1, 1, 0, 2, 0, 5, 1, 2, 2, 1, 0] se corresponde con:
    2-1, 1-0, 1-1, 0-1, **1-0, 2-0, 5-1**, 2-2, **1-0**, con una racha máxima de 3 victorias.

14. Escribe la función **NumerosPrimosPro** que nos calculará los números primos entre 1 y otro número (para probarlo, nos vale con 1000).
    Esta función usará un algoritmo más avanzado que el que hemos visto con anterioridad y que es mucho más eficiente a la hora de calcular muchos números primos.
    El funcionamiento es el siguiente (para el ejemplo del 1 al 1000):
    - Creamos un array de 1001 elementos (por el cero).
    - Empezamos en la posición 2 del array y vamos haciendo lo siguiente:
        - Si en esta posición hay un 0 (que es lo que había al principio), añadimos esta posición a una lista de números primos.
        - Si había un 0, además, tenemos que ir "tachando" todos los múltiplos de ese número primo poniendo otro valor en esas posiciones (por ejemplo, un -1). En el caso del 2, que sería el primero, pondríamos un -1 en la posición 4, 6, 8, 10, 12, etc., y así hasta llegar al 1000.
        - Si en la posición hay un -1, eso es que este número ya era múltiplo de uno anterior, así que no hacemos nada y pasamos al siguiente.
    - Al final, devolveremos la lista de números primos que hemos ido recopilando.

    Para comprobarlo, mostrar los números primos entre 1 y 1000 por pantalla y escribir cuántos números son (empieza por 2, 3, 5, 7... acaba por 983, 991, 997 y son 168 números primos en total).

15. Escribe la función **SucesionFibonacci** a la que le pasas un array de enteros vacío y nos lo rellena con los números de la sucesión de Fibonacci (hasta el tamaño del array).
    La sucesión de Fibonacci funciona así:
    - El primer valor es el 0.
    - El segundo valor es el 1.
    - Del tercer valor en adelante, cada valor es la suma de los dos anteriores.

    Así, la sucesión de Fibonacci es: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233…

    Si, por ejemplo, le pasamos un array de 5 elementos, nos los debería rellenar con los valores [0, 1, 1, 2, 3]

16. Escribe una función **SeriesMundiales**. La función recibe tres parámetros:
    - Un *string* con el nombre del equipo local
    - Un *string* con el nombre del equipo visitante
    - Un array de enteros con los resultados de los partidos de béisbol.

    En este array vienen las puntuaciones de los partidos. Para cada partido, vienen 2 números: las carreras que ha anotado el equipo de fuera y las carreras que ha anotado el equipo de casa (los americanos ponen las cosas al revés).

    Por ejemplo, en el array [4,3,3,8] tendríamos dos partidos: en el primero habría ganado el equipo visitante 4-3 y en el segundo habría ganado el equipo de casa 8-3.

    La función calculará el número de partidos que ha ganado cada equipo y devolverá el nombre del equipo ganador.

    Dará un error en el caso de que algún partido haya quedado en empate (no existe el empate en béisbol) o si el visitante y el local han ganado el mismo número de partidos (no pueden quedar empatadas las Series Mundiales). También dará error, obviamente, si el tamaño del array es impar.

    Para probarlo, este año el equipo local eran los Boston Red Sox y el visitante Los Angeles Dodgers. El array con los resultados sería: [4,8,2,8,3,2,6,9,1,5]. Ganaron los Red Sox 4-1

17. Escribe la función **ElementosComunes3** a la que le pasaremos tres arrays de enteros. La función será de tipo void y no devolverá nada.

    La función deberá imprimir por pantalla los elementos sean comunes a los tres arrays, es decir, que estén en el primero y también en el segundo y también en el tercero.

    Ej.: a1=[1, 3, 5, 7], a2=[2, 3, 4, 5, 6, 7], a3=[1, 2, 3, 4, 5]

    Escribirá por pantalla el 3 y el 5, que son los elementos comunes a los 3.

18. Escribe la función **SeparaNegativosPositivos** a la que le pasamos un array de enteros y nos devolverá otro array de enteros con el resultado.

Esta función deberá separar los números negativos y positivos que hay dentro del array, de manera que los negativos queden al principio y los positivos queden al final, pero manteniendo el orden original de éstos. El array original no debe sufrir modificaciones.

Ej.: [5, 2, -1, -2] -> [-1, -2, 5, 2]
[4, -5, 1, -8, 2, -1] -> [-5, -8, -1, 4, 1, 2]

19. Escribe la función **SumaPosicionesPares** a la que le pasamos un array de enteros y nos devuelve la suma de los valores que hay en las posiciones pares del array (o sea, la posición 0, la 2, etc.).

    Ej.: Si le pasamos [10, 20, 30, 40, 50], nos devolverá 90 (10+30+50)

20. Escribe la función **ArrayCapicua** a la que la pasamos un array y nos devuelve true si el array se lee igual hacia delante que hacia atrás (ej.: [1, 2, 3, 2, 1]). Deberá funcionar con cualquier tamaño de array y preferiblemente no creéis arrays adicionales.

21. Escribe la función **TroceaArray** a la que le pasamos tres arrays **a**, **b** y **c** de forma que el tamaño de **a** debe ser igual a la suma de los tamaños de **b** y **c** (por ejemplo, **a** puede tener tamaño 10, **b** ser de tamaño 6 y **c** de tamaño 4).

    La función copiará números de **a** a **b** mientras quepan en **b** y el resto los pondrá en **c** (es como ConcatenaArray, pero al revés).

    Ej.: a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], la función rellenaría los otros dos arrays con los valores: b = [1, 2, 3, 4, 5, 6] y c = [7, 8, 9, 10]