TAREA UT4

ESTRUCTURAS DEFINIDAS POR EL USUARIO

La tarea consiste en la implementación de varios scripts en lenguaje Javascript. Se valorará positivamente la claridad del código, el uso de las convenciones pedidas para el nombrado de ficheros, funciones, métodos, clases y variables y el uso de comentarios en el código, así como la corrección de los resultados obtenidos.

Es obligatorio implementar las funciones que se detallan en cada uno de los ejercicios, así como ceñirse a los nombres propuestos para las mismas, sus argumentos y tipos de entrada y valores de salida esperados.

Salvo que se indique lo contrario, todos los mensajes se mostrarán por la consola de depuración del navegador.

1. [1 punto] Crea un script que pida al usuario su nombre y apellidos (una única cadena). A continuación se mostrarán:
    a. El tamaño del nombre más los apellidos (sin contar espacios).
    b. La cadena en minúsculas y en mayúsculas.
    c. Que divida el nombre y los apellidos y los muestre en 3 líneas, donde ponga Nombre: / Apellido 1: / Apellido 2:
    d. Una propuesta de nombre de usuario, compuesto por el nombre en minúsculas, la inicial del primer apellido y la inicial del segundo apellido. Por ejemplo para Manuel Gómez Pérez sería manuelgp.

Función	Recibe	Devuelve	Descripción
tamanioSinEspacios(nombre)	**nombre:cadena**	**entero**	Devuelve el tamaño de la cadena que recibe como parámetro, ignorando espacios.
aMayusculas(nombre)	**nombre:cadena**	**cadena**	Devuelve la cadena que recibe como parámetro pasada a mayúsculas.
aMinusculas(nombre)	**nombre:cadena**	**cadena**	Devuelve la cadena que recibe como parámetro pasada a minúsculas.
aLineas(nombre)	**nombre:cadena**	**cadena**	Devuelve la cadena que recibe como parámetro separada en varias líneas
propuestaLogin(nombre)	**nombre:cadena**	**cadena**	Devuelve una propuesta de login basado en la cadena que recibe como parámetro.

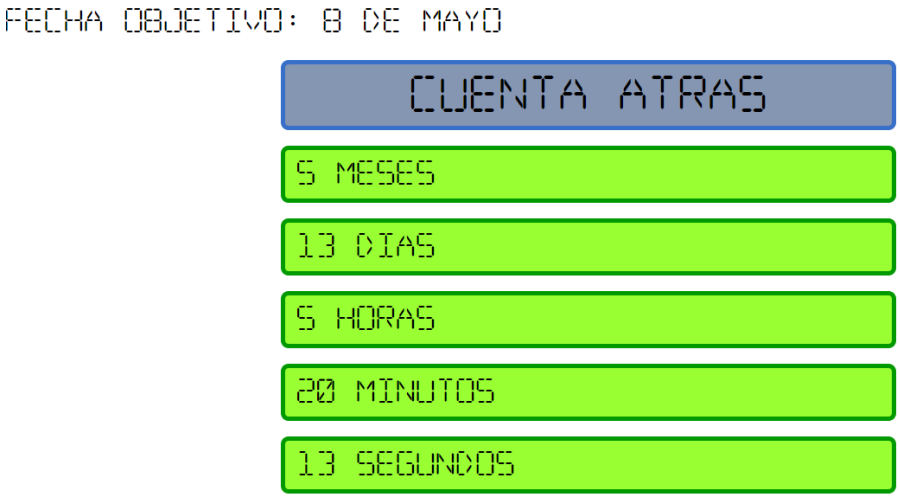
Suponemos que tanto el nombre como los apellidos son no compuestos (están formados por una única palabra).

2. [1 punto] Realiza un script que genere de manera aleatoria una cantidad de números cuyo valor varía entre 1 y 50 (la cantidad a generar también será un número aleatorio entre 500 y 1000). Dichos scripts deberán ser guardados en un array.

   Una vez generados los números, se mostrará un mensaje indicativo de cuál ha sido el número más frecuente. Para ello se contabilizarán las repeticiones de todos los números usando un array. El array tendrá 50 posiciones, y en la posición i se indicará el número de repeticiones del valor i.

Función	Recibe	Devuelve	Descripción
generaAleatorio(min,max)	**min:entero** **max:entero**	**entero**	Genera un aleatorio entre máximo y mínimo (incluidos).
generaNAleatorios(n,min,max)	**n:entero** **min:entero** **max:entero**	**entero[]**	Genera un array con n aleatorios entre máximo y mínimo.
cuentaFrecuencias(numeros)	**numeros: entero[]**	**entero[]**	Genera un array que indica, las frecuencias de repetición de cada valor contenido en **numeros**. La posición i-ésima contendrá el número de repeticiones del valor i.
maximaFrecuencia(repeticiones)	**repeticiones:entero[]**	**-**	Imprime las frecuencias de repetición de todos los elementos del array, indicando además cual es el elemento que más se repite.

3. [1 punto] Crea un script que pida al usuario la fecha de su cumpleaños en formato DD-AAAA (se repetirá la petición de datos hasta que el formato sea el pedido). A continuación se calculará el tiempo que queda para el mismo, calculando meses, días, horas, minutos y segundos (hasta las 00:00 del día indicado). La información se actualizará cada segundo:



Es obligatorio usar la plantilla HTML y el CSS proporcionado.

4. [1 punto] Quedamos almacenar los resultados obtenidos en las elecciones en Villaconejos, teniendo en cuenta que:
    - Ha habido 5 sedes para votar (Ayuntamiento, Polideportivo, Instituto, Mercado y Colegio)
    - Se han presentado 4 partidos (Puede que Villaconejos (PV), Obreros de Villaconejos (OV), Villaconejos Por el Si (VpSI), Unión Progreso y Villaconejos (UPV).

    Se generarán aleatoriamente aleatoriamente los votos correspondientes a cada partido (entre 5 y 10 votos por sede).

    A continuación se mostrarán por consola:
    - Una tabla (HTML) con todos los colegios electorales y partidos, así como sus votos asociados.
    - Calcular el número total de votos por partido y por sede
    - Indicar de mayor a menor los votos recibidos por partido.

Queremos implementar dos versiones distintas del problema en función de las estructuras de datos usadas para representar los votos:

- En una primera versión (que se almacenará en la carpeta **ejercicio3a**) mantendremos las siguientes estructuras de datos:
    - Un array con los 5 partidos.
    - Un array con las 4 sedes.
    - Una tabla tamaño 4x5 que almacena los votos, teniendo en cuenta que la posición [i][j] de la tabla almacena los votos obtenidos por el partido i en la sede j.
- En una segunda versión (que se almacenará en la carpeta **ejercicio3b**) se mantendrá una única tabla de tamaño 6x5 donde:
    - La fila 0 contendrá a los partidos.
    - La columna 0 contendrá las sedes.

Se valorará positivamente el uso correcto de funciones en la resolución del ejercicio.

5. [1 punto] Queremos hacer una aplicación en JavaScript para gestionar edificios con la información de la situación del edificio y de los propietarios de cada piso. Para ello queremos almacenar la siguiente información de cada edificio mediante una clase:
   - Calle.
   - Número.
   - Código postal.
   - Plantas del edificio (dentro de cada planta tendremos un número de puertas y para cada puerta almacenaremos el nombre del propietario). Se implementará esta propiedad utilizando un array bidimensional.

   Cada vez que instanciemos un edificio le pasaremos la calle, número y código postal como parámetros. Se pide además crear los siguientes métodos:

Clase **Edificio** [Definido mediante función constructora]			
Propiedades			
Nombre	Tipo	Visibilidad	Descripción
calle	cadena	Privada	-
numero	entero	Privada	-
codigoPostal	entero	Privada	-
plantas	cadena[][]	Privada	-
Métodos			
Nombre	Parámetros	Devuelve	Descripción
Edificio	calle:cadena numero:entero codigoPostal:entero	-	Constructor con parámetros. Deberían asignarse valores por defecto a los parámetros.
agregaPlantasYPuertas	nPlantas:entero nPuertas:entero	-	Recibe el número de plantas que queremos crear en el edificio y el número de puertas por planta. Las plantas se añadirán a las ya creadas en el edificio.
modificaNumero	numero:entero	-	Actualiza el número del edificio
modificaCalle	calle:cadena	-	Modifica el nombre de la calle.
modificaCP	cp:cadena	-	Modifica el CP
getCalle	-	cadena	Devuelve el nombre de la calle
getNumero	-	entero	Devuelve el número del edificio
getCP	-	cadena	Devuelve el CP
agregaPropietario	nombre:cadena planta:entero puerta:entero	-	Asigna el propietario **nombre** al piso identificado por **planta** y **puerta**.
imprimePlantas	-	-	Recorre el edificio e imprime por consola los propietarios de cada piso.

Cada vez que se crea un edificio que muestre por consola un mensaje del estilo:

Construido nuevo edificio en calle: xxxxxx, nº: xx, CP: xxxxx.

Cada vez que se añada un propietario a un piso de un edificio se mostrará un mensaje del estilo:

xxxxxxxx es ahora el propietario de la puerta x de la planta x.

Por otra parte se crearán varias instancias que nos permita comprobar que funciona correctamente, llamando los métodos que hemos definido.

6. [1 punto] Queremos implementar en Javascript las siguientes estructuras de datos:

Clase **Jugador**			[Definida mediante prototipo]
Propiedades			
Nombre	Tipo	Visibilidad	Descripción
nombre	cadena	Privada	-
numeroVidas	entero	Privada	-
tienePistola	booleano	Privada	-
tieneGranada	booleano	Privada	-
numeroBalas	entero	Privada	-
Métodos			
Nombre	Parámetros	Devuelve	Descripción
Jugador	nombre:cadena numeroVidas:entero tienePistola:booleano tieneGranada:booleano numeroBalas:entero	-	Constructor con parámetros. Todos los parámetros deberían inicializarse a un valor por defecto para prevenir la posibilidad de que el constructor se llame sin usarlos.
setPropiedad	propiedad:tipo	-	Establece el valor de la propiedad. Realiza las validaciones oportunas respecto a valores no vacíos y consistencia de tipos, asignando un valor por defecto en su caso.
getPropiedad	-	tipoPropiedad	Devuelve el valor de la propiedad
toString	-	Cadena	Devuelve la información correspondiente al jugador en forma de cadena. Por ejemplo: "*El jugador Pepe tiene 3 vidas, tiene pistola con 2 balas y granada*".

Clase **Partida**			[Definida mediante prototipo]
Propiedades			
Nombre	Tipo	Visibilidad	Descripción
jugadores	[ ]	Privada	-
Métodos			
Nombre	Parámetros	Devuelve	Descripción
Partida	-	-	Constructor sin parámetros. Se inicializan los datos correspondientes a 5 jugadores que se insertan en el array.
imprime	-	-	Recorre el array y muestra por consola los datos de los jugadores
ruletaRusa	-	-	Escoge al azar uno de los jugadores y establece su número de vidas a 0.

El código debería ir repartido en 3 archivos:

- **jugador.js** → Código correspondiente a la clase **Jugador**.
- **partida.js** → Código correspondiente a la clase **Partida**.
- **ejercicio6.js** → Instancia un objeto de tipo **Partida** y llama de manera secuencial a los métodos **imprime**(), **ruletaRusa**() e **imprime**() de nuevo, comprobando que funcionan correctamente.

7. [1 punto] Implementa un script que contenga la siguiente estructura de datos:

Clase **Usuario** [Definida mediante prototipos]			
Propiedades			
Nombre	Tipo	Visibilidad	Descripción
Nombre	cadena	Privada	-
Apellidos	cadena	Privada	-
Dni	cadena	Privada	-
aNacimiento	entero	Privada	-
Provincia	cadena	Privada	-
Métodos			
Nombre	Parámetros	Devuelve	Descripción
Usuario	-	-	Constructor sin parámetros. Se inicializan todas las propiedades.
Usuario	nombre:cadena apellidos:cadena dni:cadena aNacimiento:entero provincia:cadena	-	Constructor con parámetros.
setPropiedad	propiedad:tipo	-	Establece el valor de la propiedad. Realiza las validaciones oportunas respecto a valores no vacíos y consistencia de tipos, asignando un valor por defecto en su caso.
getPropiedad	-	tipoPropiedad	Devuelve el valor de la propiedad
generaLogin	-	cadena	Genera una cadena consistente en la inicial del nombre concatenada con la inicial del primer apellido y el segundo apellido, además de los dos últimos dígitos del año de nacimiento.
getEdad	-	entero	Devuelve la edad del usuario.
toString	-	cadena	Devuelve una cadena que consiste en todas las propiedades concatenadas en una única cadena sin formato.
toHTML	-	cadena	Devuelve una cadena que consiste en todas las propiedades concatenadas en una cadena en formato HTML (puede ser en forma de listado, un párrafo… como se prefiera).

Además de la implementación de la clase, en el script correspondiente:

- Se crearán 4 usuarios de prueba.
- Se mostrarán en la consola de Javascript las propiedades de cada usuario.
- Se mostrarán en la página HTML las propiedades de cada usuario.
- Se mostrará en un listado en la página HTML el DNI de cada usuario seguido de su login y su edad.

8. [1 punto] Crea una *pseudoclase* llamada `ArrayOrdenado` con las siguientes características:

Clase **ArrayOrdenado** [Definida mediante prototipos]			
Propiedades			
Nombre	Tipo	Visibilidad	Descripción
numeros	entero[]	privada	Contiene un array de enteros positivos Inicialmente vacío
orden	booleano	privada	Determina el orden en el que está ordenado el array. **true** significa ascendente y **false** significa descendente. Por defecto es orden ascendente
Métodos			
Nombre	Parámetros	Devuelve	Descripción
getPosicion(numero)	numero:entero	entero	Devuelve la posición de un número, o -1 si el número no se encuentra en el array
insertaNumero(numero)	numero:entero	entero	Inserta el número en el array, teniendo en cuenta que tiene que estar ordenado y no puede haber números repetidos. Devuelve la posición en la cual se ha insertado el número, o -1 si no se ha podido insertar (por estar repetido, o el parámetro no es entero positivo).
borraNumero(numero)	numero:entero	booleano	Borra el número que le pasamos como parámetro. Devuelve verdadero si se ha podido borrar el número y falso en caso contrario. Hay que tener en cuenta que no podemos dejar huecos en el array.
getNumeroAt(posicion)	posicion:entero	entero	Devuelve el número que está en una posición determinada, o -1 si no hay número en dicha posición..
primero()	-	entero	Devuelve el primer elemento del array.
ultimo()	-	entero	Devuelve el último elemento del array.
invierte()	-	-	Invierte el orden de los elementos del array (pasa de ascendente a descendente o viceversa). Para implementar este método se usará el método **sort()** usando una función de callback personalizada.

La clase se definirá en el archivo **arrayordenado.js**. Una vez implementada la clase se realizarán las siguientes acciones en el archivo **ejercicio8.js**:

- Se creará un objeto de tipo **ArrayOrdenado**
- Se introducirán en el mismo 100 números aleatorios cuyo valor puede ir del 1 al 100
- Se mostrará el contenido del array (recorriéndolo secuencialmente) por la consola de depuración.
- Se cambiará el orden de ordenación y se mostrará el contenido de nuevo.
- Se borrarán del array todos los números pares.
- Se volverá a mostrar el contenido del array por la consola de depuración.

9. [1 punto] Crea una *pseudoclase* llamada `ArrayUsuarios` con las siguientes características:

Clase `ArrayUsuarios` [Definida mediante prototipos]			
Propiedades			
Nombre	**Tipo**	**Visibilidad**	**Descripción**
`usuarios`	`Usuario[]`	privada	Contiene un array de usuarios. Inicialmente vacío
`orden`	`Booleano`	privada	Determina el orden en el que está ordenado el array (se sigue como criterio de ordenación el DNI del usuario). **`true`** significa ascendente y **`false`** significa descendente. Por defecto es orden ascendente.
Métodos			
Nombre	**Parámetros**	**Devuelve**	**Descripción**
`getPosicion(dni)`	`dni:cadena`	`Usuario`	Devuelve la posición de un usuario, o -1 si el usuario no se encuentra en el array
`insertaUsuario(usuario)`	`usuario:Usuario`	`entero`	Inserta el usuario en el array, teniendo en cuenta que tiene que estar ordenado y no puede haber usuarios repetidos. Devuelve la posición en la cual se ha insertado el usuario, o -1 si no se ha podido insertar

`insertaUsuario(nombre, apellidos, dni, anacimiento, provincia)`	`nombre: cadena`
	`apellidos:cadena`
	`dni:cadena`
	`aNacimiento:entero`
	`provincia:cadena`
		`entero`	Inserta el usuario con los datos que se le pasan en el array, teniendo en cuenta que tiene que estar ordenado y no puede haber usuarios repetidos.
			Devuelve la posición en la cual se ha insertado el número, o -1 si no se ha podido insertar

`borraUsuario(dni)`	`dni:Usuario`	`booleano`	Borra el usuario cuyo DNI coincide que le pasamos como parámetro. Devuelve verdadero si se ha podido borrar el usuario y falso en caso contrario. Hay que tener en cuenta que no podemos dejar huecos en el array.
`getUsuarioAt(posicion)`	`posicion:entero`	`Usuario`	Devuelve el usuario que está en una posición determinada, o null si no hay usuario en dicha posición.
`primero()`	-	`Usuario`	Devuelve el primer elemento del array.
`ultimo()`	-	`Usuario`	Devuelve el último elemento del array.
`invierte()`	-	-	Invierte el orden de los elementos del array (pasa de ascendente a descendente o viceversa). Para implementar este método se usará el método **`sort()`** usando una función de callback personalizada.

La clase se definirá en el archivo **`arrayusuarios.js`**. Una vez implementada la clase se realizarán las siguientes acciones en el archivo **`ejercicio9.js`**:

- Se creará un objeto de la clase **`ArrayUsuarios`**.
  - Se crearán 4 usuarios de prueba y se introducirán en el array.
    - Dos usuarios serán de la provincia "Asturias" y dos serán de la provincia "Cantabria"
    - Dos usuarios serán mayores de edad y dos menores de edad.
- Se mostrará el contenido del array (recorriéndolo secuencialmente) en la página HTML.
- Se mostrarán los usuarios de Asturias ordenados por DNI ascendentemente.
- Se mostrarán los usuarios de Cantabria ordenados por DNI descendientemente.

10. [1 punto] Modifica la clase **`ArrayOrdenado`** mediante herencia para permitir que en la clase se almacenen números duplicados (llama a la nueva clase **`ArrayOcurrencias`**). Para ello se tendrán en cuenta las siguientes consideraciones:
    - Se añadirá un nuevo array que representará las ocurrencias de cada número.
    - El número se eliminará realmente cuando haya una única ocurrencia del mismo. En caso contrario se actualizará el número de ocurrencias.
    - Se implementará el método `getNumeroOcurrencias(numero):numero`, que devuelve el número de ocurrencias relativas a determinado número (y -1 si el número no está en el array).

    Ten en cuenta que algunos métodos se podrán reutilizar directamente, otros habrá que reescribirlos completamente y otros se podrán reutilizar en parte (en primer lugar se llamará al método de la clase base y luego se implementará la ampliación).

La clase se definirá en el archivo **`arrayduplicados.js`**. Una vez implementada la clase se realizarán las siguientes acciones en el archivo **`ejercicio10.js`**:

- Se creará Un objeto de tipo `ArrayOcurrencias`.
- Se introducirán en el mismo 100 números aleatorios cuyo valor puede ir del 1 al 50.
- Se mostrará el contenido del array (recorriéndolo secuencialmente) por consola. Para cada posición del array se mostrará el número contenido en el mismo y el número de ocurrencias.
- Se ordenara el array en orden descendente y se mostrará el contenido del mismo.
- Se borrarán del array todos los números impares. Queremos que el borrado se realice de manera efectiva.
- Se volverá a mostrar el contenido del array por la consola de depuración.