UD03. RETOS MISION II VIRUS SANDIEGO

1. u2e1_string.html

(Este ejercicio está compuesto por tres partes que encontrarás en tres países diferentes. Puedes irlo completando a medida que vayas encontrando su enunciado).

Diseña un programa que solicite a un usuario en una sola petición su nombre, dos apellidos y año de nacimiento (por ejemplo: «Pepe Perez Sanchez 1990». El nombre y los apellidos serán una única palabra cada uno, pero no hace falta que lo valides. No habrá Juan Tomás o San Salvador.Sobre ese nombre deberá mostrar la siguiente información para crear un nombre de usuario:

- Transformación de la cadena a todos mayúsculas.
- Muestra de nombre, apellido1, apellido2 y año, cada cuál en una línea.
- Nombre de usuario: estará formado por la inicial del nombre, el primer apellido, la inicial del segundo apellido y las dos últimas cifras del año de nacimiento, todo en minúsculas (ej. pperezs90).

2. u2e1_string.html

(Este ejercicio está compuesto por tres partes que encontrarás en tres países diferentes. Puedes irlo completando a medida que vayas encontrando su enunciado).

Diseña un programa que solicite a un usuario en una sola petición su nombre, dos apellidos y año de nacimiento (por ejemplo: «Pepe Perez Sanchez 1990». El nombre y los apellidos serán una única palabra cada uno, pero no hace falta que lo valides. No habrá Juan Tomás o San Salvador.

Sobre ese nombre deberá mostrar la siguiente información para crear un nombre de usuario:

- Tamaño de la cadena que contiene, únicamente, nombre y dos apellidos.

Continuará…

3. u2e1_string.html

(Este ejercicio está compuesto por tres partes que encontrarás en tres países diferentes. Puedes irlo completando a medida que vayas encontrando su enunciado).

Diseña un programa que solicite a un usuario en una sola petición su nombre, dos apellidos y año de nacimiento (por ejemplo: «Pepe Perez Sanchez 1990». El nombre y los apellidos serán una única palabra cada uno, pero no hace falta que lo valides. No habrá Juan Tomás o San Salvador.Sobre ese nombre deberá mostrar la siguiente información para crear un nombre de usuario:

- Número de vocales que contiene la cadena de nombre y apellidos.

Continuará…

4. u2e2_number.html

Crea un conversor de bases que solicite al usuario en qué base está el número que va a introducir (binario, decimal, octal, hexadecimal) y muestre el número en las 4 bases. Trata de hacerlo de la manera más optimizada posible.

5. U2e3_math.html

Realiza un programa que permita elegir al usuario un tipo de lotería del estado y genere os números aleatorios en la propia página (no en un alert). Se mostrarán los siguientes tipos de loterías:

- Lotería nacional (los ceros a la izquierda deben mostrarse).
- Quiniela.
- Primitiva (con complementario y reintegro).

6. u2e4_date1.html

Realiza un programa que sea capaz de calcular cuántas veces cae en sábado el solsticio de verano desde el año 2000 hasta el año 2100.

7. u2e4_date2.html

Realiza un archivo en js con tres funciones que te permitan realizar los siguientes formatos de fecha recibiendo como parámetro la fecha en milisegundos o en números (3 parámetros) -utiliza arguments para saber el número de parámetros que ha introducido el usuario-:

- Fecha en formato corto: 17/02/2016
- Fecha en formato largo: Miércoles, 17 de febrero de 2016.
- Fecha en formato inglés: Wednesday, February 17, 2016.

8. u2e4_date3.html

Completa el archivo con dos funciones más que te permitan mostrar la hora en los siguientes formatos:

- Hora en formato corto: 14:30.
- Hora en formato largo: 14:30:25.
- Hora con PM/AM: 02:30 AM si es antes de medio día o 02:30 PM si es después del mediodía. No se utilizan horas mayores que 12 ni menores que 1 (no existe 00:30).

9. u2e5_window.html

Realiza un programa que tenga una ventana principal, con tres botones:

- Abrir: permitirá abrir una pequeña ventana a modo de pop-up. Esta ventana secundaria no tendrá ni barras, ni scroll, ni permita que se redimensione, y tendrá

el tamaño necesario para los elementos que contenga (elige tú las medidas).
- Cerrar: permitirá cerrar la ventana creada.
- Imprimir: permitirá imprimir la propia ventana.

En la ventana secundaria (pop-up) habrá al menos, los siguientes botones:

- 6 botones con nombres de colores. Cuando se pulse cualquiera de los botones se imprimirá en la pantalla principal un párrafo con el color indicado en el botón (puedes elegir colores predefinidos como red, blue, green… o poner códigos hexadecimales de color).
- Un botón para cerrar la propia ventana.
- Un botón para colocar la ventana en otra localización. Para ello mostrará un mensaje para que el usuario indique, mediante dos valores separados por coma, la posición desde arriba y desde la izquierda.

10. u2e6_windowtiempo.html

Necesitas un reloj con cronómetro y lo vas a desarrollar en Javascript. El reloj mostrará la hora, con minutos y segundos. El cronómetro, a su vez, tendrá lo siguiente:

- Un botón para poner a 0 el contador.
- Un botón para arrancar el cronómetro.
- Un botón para parar el cronómetro.
- Un botón para escribir el valor del cronómetro parado en la página (podrán escribirse tantos valores como se desee).

11. u3e1_objetos.html

Crea tres objetos que te permitan almacenar información de sandkills. Deberás realizarlos de la siguiente manera.

- sandkill1: creado como un literal.
- sandkill2: creado con la definición de Object (utilizando new).
- sandkill3: primero crearás un constructor de objeto llamado Sandkill y a partir de él te crearás el objeto sandkill3.

Los tres objetos tendrán las mismas propiedades: nombre, edad, especialidad. Y un método .mostrar() que devuelva una cadena con la información de cada sandkill en una sola línea.

Posteriormente añade a sandkill1 la propiedad «nacionalidad» y a sandkill2 la propiedad «lenguajeFavorito». Por último, borra de sandkill3 la propiedad «especialidad».

Crea una función que, pasado un objeto de tipo Sandkill, muestre todas sus propiedades, pero sin llamar a su método mostrar. Llama a la función tres veces para mostrar la información de los tres objetos creados.

Modifica en sandkill1 el nombre, en sandkill2 el lenguajeFavorito y en sandkill3 la edad. Vuelve a llamar a la función de mostrar las propiedades de los tres objetos.

12. u3e2_prototipos.html

Crea un buen prototipo para almacenar información de Sandkills. Los objetos de este tipo tendrán la siguiente información:

- Nombre.
- Edad.
- Especialidad: almacenará un número (1, 2, 3).
- Compañero: almacenará un objeto de tipo Sandkill.

Además dispondrá de los siguientes métodos:

- mostrar: devuelve una cadena con la información del Sandkill nombre, edad, especialidad y nombre del compañero. ¡Ojo! Para mostrar esta información utilizarás los métodos get de cada propiedad.
- getNombre: devuelve el nombre del sandkill.
- getEdad: devuelve la edad del sandkill.
- getEspecialidad: devuelve una cadena con la especialidad a partir del número indicado. Si es 1, devuelve Sistemas; si es 2 devuelve Web; si es 3 devuelve Multiplataforma.
- getNombreCompanero: devuelve una cadena con el nombre del compañero (getNombre).
- getCompanero: devuelve un objeto de tipo Sandkill.
- setNombre: modifica el nombre del sandkill.
- setEdad: modifica la edad del sandkill.
- setEspecialidad: modifica la especialidad del sandkill (recibe un número).
- setEspecialidadNombre: recibe una cadena y almacena un número en función de si la cadena recibida es Sistemas (1), Web (2) o Multiplataforma(3).
- setCompanero: recibe un objeto de tipo Sandkill y lo almacena.

13. u3e3_arrays.html

El siguiente programa necesitará dos archivos en javascript. El primero contendrá el prototipo Sandkill creado en el ejercicio anterior. El segundo permitirá trabajar con arrays de cualquier tipo de dato.

Diseña un programa que permita realizas las siguientes operaciones:

- Insertar un sandkill en una lista al principio.
- Insertar un sandkill en una lista al final.
- Borrar el primer sandkill de la lista.
- Borrar el último sandkill de la lista.
- Mostrar la lista de sandkills.

- Mostrar la lista de sandkills ordenada.
- Buscar un sandkill a partir de su nombre.
- Buscar un sandkill a partir de su posición.

Hazlo de tal manera que puedas reutilizar el archivo de gestión de arrays y te funcione con cualquier tipo de objeto que crees.

14. u3e4_prueba_final_electricidad.html

Diseña un programa que permita la gestión completa de los siguientes datos:

- Escuela: contendrá la información de las escuelas (nombre, localidad, responsable…).
- Instalacion: contendrá información de las instalaciones eléctricas que puede necesitar una escuela (tipo -luz exterior, luz interior, enchufes-, cantidad, presupuesto).
- Compania: contendrá la información de las compañías eléctricas del país (nombre, responsable…).

Ten en cuenta que una escuela puede tener varias instalaciones (solo una de cada tipo, por tanto, máximo 3), y cada instalación la puede realizar una compañía diferente. Por tanto, elegid la estructura de objetos más adecuada para su correcto almacenamiento.

ESPECIFICACIONES:

- El programa debe ser capaz de gestionar correctamente todos los elementos, con su correspondiente inserción, modificación, borrado, visualización, búsqueda, etc.
- Cada miembro del equipo debe desarrollar uno de los objetos, pero los tres deben estar integrados en el mismo programa, por lo que es conveniente que os pongáis de acuerdo a la hora de estructurar el código de modo que sea lo más homogéneo posible.
- El código de cada objeto debe encontrarse en un archivo .js independiente. El código de gestión de objetos también. Además, debe haber un archivo HTML que permita interactuar con el usuario.
- Todo el código debe estar comentado y se debe incluir el JSDoc correspondiente. RECUERDA QUE NUNCA SE SABE QUIÉN VA A TENER QUE MANTENERLO.

15. u3e4_prueba_final_hospital.html

Diseña un programa que permita la gestión completa de los siguientes datos:

- Hospital: contendrá la información del hospital (nombre, localidad, responsable…).
- Personal: contendrá información del personal del hospital (nombre, especialidad -médico, enfermera, celador-, etc.
- Paciente: contendrá la información de los pacientes de un hospital. Cada paciente tendrá asignado un Personal..

Ten en cuenta que un hospital puede tener mucho personal, y muchos pacientes. Suponemos que un paciente y un personal solo pueden estar en un hospital. Por tanto,

elegid la estructura de objetos más adecuada para su correcto almacenamiento.

ESPECIFICACIONES:

- El programa debe ser capaz de gestionar correctamente todos los elementos, con su correspondiente inserción, modificación, borrado, visualización, etc.
- Cada miembro del equipo debe desarrollar uno de los objetos pero los tres deben estar integrados en el mismo programa, por lo que es conveniente que os pongáis de acuerdo a la hora de estructurar el código de modo que sea lo más homogéneo posible.
- El código de cada objeto debe encontrarse en un archivo .js independiente. El código de gestión de objetos también. Además, debe haber un archivo HTML que permita interactuar con el usuario.
- Todo el código debe estar comentado y se debe incluir el JSDoc correspondiente. RECUERDA QUE NUNCA SE SABE QUIÉN VA A TENER QUE MANTENERLO.

16. u3e4_prueba_final_escuela.html

Diseña un programa que permita la gestión completa de los siguientes datos:

- Escuela: contendrá la información de las escuelas (nombre, localidad, responsable…).
- Profesor: contendrá información de los profesores que trabajan allí (nombre, tipo - ciencias, letras-…).
- Alumno: contendrá la información de los alumnos de la escuela (nombre, curso, profesor responsable).

Ten en cuenta que una escuela puede tener varios profesores y alumnos. No hay un alumno que vaya a dos escuelas diferentes, ni dos profesores que trabajen en distintas escuelas. Por tanto, elegid la estructura de objetos más adecuada para su correcto almacenamiento.

ESPECIFICACIONES:

- El programa debe ser capaz de gestionar correctamente todos los elementos, con su correspondiente inserción, modificación, borrado, visualización, búsqueda, etc.
- Cada miembro del equipo debe desarrollar uno de los objetos, pero los tres deben estar integrados en el mismo programa, por lo que es conveniente que os pongáis de acuerdo a la hora de estructurar el código de modo que sea lo más homogéneo posible.
- El código de cada objeto debe encontrarse en un archivo .js independiente. El código de gestión de objetos también. Además, debe haber un archivo HTML que permita interactuar con el usuario.
- Todo el código debe estar comentado y se debe incluir el JSDoc correspondiente. RECUERDA QUE NUNCA SE SABE QUIÉN VA A TENER QUE MANTENERLO.

17. u3e4_prueba_final_tanques.html

Diseña un programa que permita la gestión completa de los siguientes datos:

- Tanque: contendrá la información de cada tanque (número, capacidad, localidad…)
- Localidad: almacenará información de localidades (nombre, número de habitantes, provincia…).
- Habitante: contendrá información de los habitantes que pueden acceder a los tanques para abastecerse de agua (nombre, edad, localidad…).

Ten en cuenta que los números de tanque son únicos. Que una localidad puede tener varios tanques. Y que un habitante únicamente puede acceder a un tanque. Por tanto, elegid la estructura de objetos más adecuada para su correcto almacenamiento.

ESPECIFICACIONES:

- El programa debe ser capaz de gestionar correctamente todos los elementos, con su correspondiente inserción, modificación, borrado, visualización, búsqueda, etc.
- Cada miembro del equipo debe desarrollar uno de los objetos, pero los tres deben estar integrados en el mismo programa, por lo que es conveniente que os pongáis de acuerdo a la hora de estructurar el código de modo que sea lo más homogéneo posible.
- El código de cada objeto debe encontrarse en un archivo .js independiente. El código de gestión de objetos también. Además, debe haber un archivo HTML que permita interactuar con el usuario.
- Todo el código debe estar comentado y se debe incluir el JSDoc correspondiente. RECUERDA QUE NUNCA SE SABE QUIÉN VA A TENER QUE MANTENERLO.

18. u3e4_prueba_final_biblioteca.html

Diseña un programa que permita la gestión completa de los siguientes datos:

- Biblioteca: contendrá la información de la biblioteca(nombre, localidad, responsable…).
- Libro: contendrá información de los libros de la biblioteca (titulo, autor, prestado -contendrá un socio-).
- Socio: contendrá la información de los socios de la biblioteca (nombre, biblioteca…).

Ten en cuenta que una biblioteca puede tener varios libros y varios socios. Un socio solo puede serlo de una biblioteca. Un libro podrá estar prestado a un socio. Y un libro solo está en una biblioteca (no tenemos en cuenta que pueda haber el mismo libro en varias bibliotecas).

ESPECIFICACIONES:

- El programa debe ser capaz de gestionar correctamente todos los elementos, con su correspondiente inserción, modificación, borrado, visualización, búsqueda, etc.
- Cada miembro del equipo debe desarrollar uno de los objetos pero los tres deben estar integrados en el mismo programa, por lo que es conveniente que os pongáis de acuerdo a la hora de estructurar el código de modo que sea lo más homogéneo posible.
- El código de cada objeto debe encontrarse en un archivo .js independiente. El código

de gestión de objetos también. Además, debe haber un archivo HTML que permita interactuar con el usuario.

- Todo el código debe estar comentado y se debe incluir el JSDoc correspondiente. RECUERDA QUE NUNCA SE SABE QUIÉN VA A TENER QUE MANTENERLO.