GUIÓN DE ACTIVIDAD

DESARROLLO WEB EN ENTORNO CLIENTE

ACTIVIDAD RECUPERACIÓN: EJERCICIOS JAVASCRIPT II

OBJETIVOS

- Trabajar con elementos avanzados de funciones.
- Trabajar con funciones flecha. Ámbitos de funciones.
- Trabajar con funciones de primer orden.
- Trabajar con funciones de orden superior.
- Funciones de orden superior del tipo array.
- Conocer y usar objetos en Javascript.

TEMPORALIZACIÓN

Aproximadamente 3-4 horas

PROCESO DE DESARROLLO

1. Dado el siguiente conjunto de valores: “Ana”, 2, “Javi”, 5, “Roberto”, 7,“Vanessa”, 10:
    a. Escribe una función, sin el uso de bucles, que elimine los valores que no sean te tipo texto y devuelva un nuevo conjunto con el resultado y lo muestre por consola **(1 punto)**
    b. Escribe una función que filtre los valores que no sean impares o no sean de tipo numérico y devuelva un nuevo array con el resultado y lo muestre por consola. **(1 punto)**

2. Crea una función llamada “average” que tome por parámetro un array llamado “dataArray” y calcule la media de los valores almacenados en el array. La función deberá devolver la media de dicho conjunto o undefined en caso de que el array no tenga elementos. Emplea funciones de orden superior sobre arrays (e.g., map, filter, reduce, etc.) **(1.5 puntos)**
3. Crea una función llamada “findMinimum” que tomo un array numérico llamado “values” como parámetro, y devuelva el valor mínimo encontrado el dicho array. Emplea la función de orden superior sobre arrays reduce. **(1.5 puntos)**
4. Crea una función llamada “findGreaterThan” que tome por parámetro un número x, y un array de datos “values”. La función devolverá cierto en caso de

que TODOS los elementos sean mayores que x, y falso en caso contrario. **(1.5 puntos)**

5. Crea una función llamada “multipleFactorial” que tome como parámetro un array de número llamado “values”, y devuelva un nuevo array que sea el resultado de calcular el factorial para cada uno de los elementos en el array. Emplea funciones de orden superior (map, filter, reduce, etc…)**(1.5 puntos)**
6. Crea una función que tome un array de nombres de usuario llamado “users”, y un array de nombres de usuarios baneados llamado “blackListed”, y que devuelva un nuevo array con los usuarios no baneados en el array inicial. Emplea funciones de orden superior **(2 puntos)**

EVALUACIÓN

En la parte de actividades prácticas de la evaluación. Esta actividad tendrá un peso del 20%.

OBSERVACIONES

Se valorará positivamente la eficiencia y el uso de un menor número de líneas de código (siempre que la legibilidad no se vea afectada)

Deberá haber **al menos 1** commit de código por cada ejercicio terminado, indicando como mensaje del commit: “AE1-N”, donde N es el número de ejercicio. Cada ejercicio de la actividad se debe desarrollar en 1 rama dedicada.

La entrega de la actividad debe ser el zip con todo el código. El nombre del fichero zip de seguir el siguiente formato:

<APELLIDOS>__<NOMBRE>__<AEN>.zip, donde N es el número de la actividad. Por ejemplo:

SERRANO_PONS__FRANCISCO__AE1.zip

La entrega en formato incorrecto penalizará la calificación hasta en un 20%.

Cada ejercicio se puede desarrollar sobre un fichero diferente llamado “ae1_N.js”.

RÚBRICA

Criterio	No/nunca	Poco/a veces	Moderado / regularmente	Mucho / frecuentemente	Sí/siempre
Funcionalidad: cumple con las especificaciones requeridas en el enunciado. **Valor máximo: 50%**	No cumple ninguno de los requerimientos. **Valor: 0%**	Cumple parte de los requerimientos, pero falla en lo principal. **Valor: 7%**	Cumple parte de los requerimientos esenciales, pero está incompleto. **Valor: 15%**	Cumple los requerimientos esenciales, pero se escapa algún requerimiento menor. **Valor: 30%**	Cumple todos los requerimientos especificados. **Valor: 70%**

Operatividad: el desarrollo es operativo. Compila y está libre de errores en tiempo de compilación/ejecución. **Valor máximo: 25%**	No compila / no se ejecuta. **Valor: 0%**	Compila, pero sufre errores tempranos en ejecución. **Valor: 5%**	Compila y no da errores durante la fase inicial de ejecución. **Valor: 10%**	No lanza errores más que en alguna funcionalidad menor. **Valor: 15%**	No falla. Libre de errores de principio a fin de la ejecución. **Valor: 10%**
Legibilidad: el código es legible, inteligible y está bien estructurado. Las nomenclaturas de elementos son consistentes a lo largo de la aplicación. **Valor máximo: 10%**	El código es prácticamente ilegible. Código "espagueti" sin cumplir normas de formato, indentación ni exención de "código muerto". **Valor: 0%**	El código es poco legible. En general, mal formateado, y con presencia de partes de código muerto y/o comentarios evitables. Formato poco consistente. **Valor: 2%**	El código se puede leer, pero hay fallos de formateo. No hay código muerto ni comentarios que se puedan obviar. **Valor: 5%**	Buena legibilidad de código. No existe código muerto y, en general, se aprecia un buen formateo de código. Pocos comentarios obvios. **Valor: 10%**	Perfecta legibilidad de código. No existe código muerto. Formateo de código. Sin comentarios obvios. **Valor: 20%**