EJERCICIOS JAVASCRIPT (I)

1. Implementar el control de 3 variables (`a`, b, `c`) para que se muestre un mensaje de error cuando se produzca alguna de las siguientes situaciones:

a) Al menos una de las 3 variables es negativa

b) Las tres variables son iguales a 0

c) Las suma de las 3 variables es mayor que 10 Y las tres variables son diferentes

2. Diseñar una función que acepta un argumento `x` y que muestre los siguientes mensajes:

a) 'Este es muy fácil… ¡prueba otro!', si `x` es 0

b) 'El número es par', si `x` es 2, 4 ó 6

c) 'El número es impar', si `x` es 1, 3 ó 5

d) '¡¡Sólo sé contar de 0 a 6!!', para los demás casos

3. Realizar un bucle `for` que actualice una variable `i` y otra `j` de la siguiente forma:

1) `i` comienza en 0, j comienza en 20

2) El bucle debe parar cuando `i` sea mayor que 8 ó `j` sea menor que 0

3) `i` se incrementa de 1 en 1, `j` se decrementa de 3 en 3

4) Dentro del bucle sólo puede estar la sentencia `console.log(i, j)`

4. En el siguiente bucle `while` tenemos que tener cuidado porque hay operaciones que pueden dar error (no puede haber raíces cuadradas de números negativos, ni divisiones por cero).

```
let result = 1, arg1, arg2, res1, res2;
while (result > 0) {
  arg1 = Math.random() - 0.5;
  res1 = Math.sqrt(arg1);
  arg2 = Math.random();
  res2 = res1/(arg1 + arg2);
  result = res1 + res2;
  console.log(result);
}
```

a) Cambia el bucle `while` usando `continue` y `break` para que si hay una raíz cuadrada de un número negativo directamente pase a la siguiente iteración. En el caso de haber una división por cero, el bucle debe detenerse inmediatamente.

b) Prueba a realizar el apartado anterior sin usar ni `continue` ni `break`. ¿Es más sencillo?

c) Cambia el bucle `while` original para que se ejecute sin tener que asignar un valor inicial a la variable `result`.

5. Muestra la fecha y hora actuales en formato: "DD/MM/YYYY hh:mm:ss TimeZone"

6. Construye una fecha a partir del texto “02/04/2015”. Muestra la fecha con console.log, ¿qué obtienes? Prueba ahora con “2015-04-02”.

¿Cómo podrías crear una fecha con el texto “April 02, 2015”?

7. ¿Cuántos segundos han pasado desde que naciste? ¿y minutos? ¿y horas? ¿y días? Implementar una función que dada una fecha cualquiera en formato “yyyy-mm-dd”, devuelva cuánto tiempo ha pasado desde esa fecha. La función aceptará un segundo parámetro para indicar en qué unidad se quiere obtener el resultado: ‘d’ → días, ‘h’ → horas, ‘m’ → minutos y ‘s’ → segundos. Si no es ninguna de estas unidades, se mostrará un error.

8. Implementar una función `hdec2hms(x)` que transforme una hora en formato decimal a su equivalente en formato `hh:mm:ss` (en texto). Por ejemplo, 8.25 = “8:15:00” y 10.12 = “10:07:12”. Implementar también la función inversa `hms2hdec(h,m,s)`.

9. Como hemos visto, javascript, como muchos otros lenguajes, utilizan el “UNIX EPOCH” que cuenta el tiempo a partir de la medianoche del 1 de enero de 1970. Sin embargo, hay otros tipos de fecha que tienen otro origen del tiempo. Por ejemplo, la **fecha juliana** empieza a contabilizar el tiempo desde las 12:00 (mediodía) del 1 de enero de 4713 a.C., e indica el número de días (y fracciones) que han transcurrido desde ese momento. La fecha juliana se suele utilizar para calcular el tiempo que ha transcurrido desde eventos que sucedieron en la antigüedad y es aún ampliamente usada para fenómenos astronómicos e históricos lejanos. En la wikipedia (https://es.wikipedia.org/wiki/Fecha_juliana) se puede encontrar la relación entre fecha juliana y tiempo Unix (contado a partir desde 1/1/1970), siendo esta:

`fecha_juliana = tiempo_unix + 2440587.5`

Implementar una función que dada una fecha en texto (por ejemplo “`2018-09-20`”), devuelva la fecha juliana equivalente. Para realizar la suma, ten en cuenta que la fecha juliana almacena días, mientras que el tiempo_unix en javascript contiene milisegundos, por lo que hay que transformarlo en días. Por cierto, ¿de dónde crees que ha salido el valor 2440587.5?

10. Cuando hablamos de ángulos, normalmente utilizamos los grados sexagesimales (se suelen representar como “deg”, una circunferencia completa son 360º deg). Sin embargo, en trigonometría es más común utilizar los radianes (“rad”, una circunferencia completa son 2π rad). De esta forma, 360º equivalen a 2π. Las funciones trigonométricas de javascript, como la mayoría de otros lenguajes, trabajan en radianes, sin embargo en muchas ocasiones nosotros vamos a querer trabajar en grados. Por ello, se pide:

a) Implementar una función llamada `deg2rad(x)` que transforme de grados a radianes, y su inversa `rad2deg(x)`

b) Implemtentar una función `sinDeg(x)` que devuelva el seno del ángulo `x` y otra función `cosDeg(x)` que devuelva el coseno del ángulo `x` (en ambos casos `x` está en grados).

c) El seno y el coseno están relacionados por la siguiente ecuación: $\sin^2 x + \cos^2 x = 1$. Utilizar esta relación para implementar una función `sinDegAlt(x)` que obtenga el seno de un ángulo `x` (en grados) usando para ello la función `cosDeg(x)` anteriormente implementada. Implementar también la función inversa al apartado anterior: `cosDegAlt(x)` usando para ello `sinDeg(x)`