MODO ESTRICTO

El modo estricto de ECMAScript 5 es una forma de elegir una variante restringida de JavaScript, así implícitamente se deja de lado el modo poco riguroso. El modo estricto no es sólo un subconjunto: intencionalmente tiene diferencia semántica del código normal. Los navegadores que no admiten el modo estricto ejecutarán el código con un comportamiento diferente a los que sí lo soportan, por lo tanto, no confíes en el modo estricto sin antes hacer pruebas de sus características más relevantes. Los modos estricto y no estricto pueden coexistir, el código se puede transformar a modo estricto incrementalmente.

El modo estricto tiene varios cambios en la semántica normal de JavaScript:

- Elimina algunos errores silenciosos de JavaScript cambiándolos para que lancen errores.
- Corrige errores que hacen difícil para los motores de JavaScript realizar optimizaciones: a veces, el código en modo estricto puede correr más rápido que un código idéntico pero no estricto.
- Prohíbe cierta sintaxis que probablemente sea definida en futuras versiones de ECMAScript.

Para invocar el modo estricto en todo un script, escribe exactamente "use strict"; (o 'use strict';) antes de cualquier otra expresión.

```
// Sintaxis del modo estricto para todo el script
"use strict";
var v = "¡Hola! ¡Estoy en modo estricto para script!";
```

Concatenar scripts no produce problemas si todos están en modo estricto (o si todos están en modo no estricto). El problema es mezclar scripts en modo estricto con scripts en modo no estricto. Por eso se recomienda habilitar el modo estricto a nivel de función solamente (al menos durante el periodo de transición de un programa).

1.1.- MODO ESTRICTO PARA FUNCIONES

De igual forma, para invocar el modo estricto para una función, escribe exactamente "use strict"; (o 'use strict';) en el cuerpo de la función antes de cualquier otra expresión.

```
function strict() {
  // Sintaxis del modo estricto a nivel de función
  "use strict";
  function nested() {
    return "¡Y yo también!";
  }
  return "¡Hola!  ¡Soy una función en modo estricto!  " + nested();
}
function notStrict() {
  return "Yo no soy estricto.";
}
```

1.2.- MODO ESTRICTO PARA MÓDULOS

ECMAScript 2015 introdujo módulos y por tanto una tercera manera de entrar en el modo estricto. Todo el contenido de los módulos de JavaScript se encuentra automáticamente en modo estricto, sin necesidad de una declaración para iniciarlo.

```
function strict() {
  // debido a que este es un módulo, soy estricto por omisión
}
export default strict;
```

Para ampliar información sobre el modo de funcionamiento estricto en distintos ejemplos consulta la siguiente web https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Strict_mode

OBJETOS PREDEFINIDOS

2.1.- VISIÓN GENERAL Y TIPOS.

JS incluye una serie de objetos predefinidos que podemos utilizar a la hora de programar un cliente web. Cada uno de estos objetos contará con propiedades y métodos específicos y a menudo se llaman objetos nativos del lenguaje.
Los objetos nativos podemos agruparlos en 3 tipos según su función:

- Tipo 1: al igual que en Java los tipos de datos primitivos cuentan con una clase genérica, que en Java llamábamos wrapper (o envoltorio). La función de estas clases es dotar a las variables en nuestros scripts de métodos diversos. De esta forma en JS tenemos los siguientes objetos nativos: Object, Number, Boolean, String, Array, Function, Date, RegExp, Math, Symbol, Array, Set, etc.
- Tipo 2: existe un segundo tipo de objetos nativos que tienen que ver con los elementos existentes dentro de una página web y con el propio cliente web (el navegador). Son objetos utilizados para la interacción con el navegador:
  - **window**: representa la ventana del navegador y proporciona métodos y propiedades para interactuar con ella.
  - **document**: representa el documento HTML cargado en la ventana del navegador y proporciona métodos y propiedades para interactuar con él.
  - **location**: representa la URL actual de la ventana del navegador y proporciona métodos y propiedades para interactuar con ella.
  - **history**: representa el historial de navegación de la ventana del navegador y proporciona métodos y propiedades para interactuar con él.
  - **navigator**: proporciona información sobre el navegador que se está utilizando, como el nombre, la versión y el sistema operativo.
  - **screen**: proporciona información sobre la pantalla del usuario, como su tamaño y resolución.
- Tipo 3: Se trata de objetos para la gestión de errores: Error, EvalError, InternalError, RangeError, ReferenceError, SyntaxError, TypeError, URIError

2.2.- USO DE OBJETOS PREDEFINIDOS.

JS no es un lenguaje diseñado para específicamente para la programación orientada a objetos. Sin embargo es un lenguaje que soporta objetos, y hemos visto en el apartado anterior los objetos predefinidos o nativos. Para utilizar estos objetos se emplea el operador new seguido del nombre:

```
var obj = new Object(); // creamos una nueva instancia de Object.
let fecha = new Date(); // creamos una nueva instancia de Date.
```

Por otra parte para el acceso a los atributos (propiedades) y métodos utilizamos el operador punto:

```
let fecha = new Date(); // creamos una nueva instancia de Date.
alert("Dia: " + fecha.getDay());
```

Existen sin embargo algunos objetos que ya aparecen instanciados (sería el equivalente en Java a las clases estáticas). Es el caso del objeto Math. **Con estos objetos no se usa el operador new**.

```
let valor = Math.sqrt(35); // valor almacena la raíz cuadrada de 35.
```

2.3.- MATH

Es un objeto incorporado, por lo que no es necesario declarar una nueva instancia del objeto para invocar sus propiedades o métodos.
A diferencia de los demás objetos globales, el objeto Math no se puede editar. Todas las propiedades y métodos de Math son estáticos. Usted se puede referir a la constante pi como Math.PI y puede llamar a la función seno como Math.sin(x), donde x es el argumento del método. Las constantes se definen con la precisión completa de los números reales en JavaScript.

2.3.1.- PROPIEDADES DE MATH

- **Math.E**: Constante de Euler, la base de los logaritmos naturales, aproximadamente 2.718
- **Math.LN2**: Logaritmo natural de 2, aproximadamente 0.693
- **Math.LN10**: Logaritmo natural de 10, aproximadamente 2.303
- **Math.LOG2E**: Logaritmo de E con base 2, aproximadamente 1.443
- **Math.LOG10E:** Logaritmo de E con base 10, aproximadamente 0.434
- **Math.PI:** Constante PI, aproximadamente 3.14159.
- **Math.SQRT1_2:** Raíz cuadrada de ½, aproximadamente 0.707.
- **Math.SQRT2:** Raíz cuadrada de 2, aproximadamente 1.414.

2.3.2.- MÉTODOS DE MATH

Tenga en cuenta que las funciones trigonométricas (`sin()`, `cos()`, `tan()`, `asin()`, `acos()`, `atan()`, `atan2()`) devuelven ángulos en radianes. Para convertir radianes a grados, divida por (`Math.PI / 180`), y multiplique por esto para convertir a la inversa.
Tenga en cuenta que muchas de las funciones matemáticas tienen una precisión que es dependiente de la implementación. Esto significa que los diferentes navegadores pueden dar un resultado diferente, e incluso el mismo motor de JS en un sistema operativo o arquitectura diferente puede dar resultados diferentes.

- **Math.abs(x):** Devuelve el valor absoluto de un número.
- **Math.acos(x):** Devuelve el arco coseno de un número.
- **Math.acosh(x):** Devuelve el arco coseno hiperbólico de un número.
- **Math.asin(x):** Devuelve el arco seno de un número.
- **Math.asinh(x):** Devuelve el arco seno hiperbólico de un número.
- **Math.atan(x):** Devuelve el arco tangente de un número.
- **Math.atanh(x):** Devuelve el arco tangente hiperbólico de un número.
- **Math.atan2(y, x):** Devuelve el arco tangente del cociente de sus argumentos.
- **Math.cbrt(x):** Devuelve la raíz cúbica de un número.

- **Math.ceil(x):** devuelve el entero más pequeño mayor o igual a un número dado.
- **Math.clz32(x):** Devuelve el número de bits significativos de valor cero en la representación binaria de 32 bits de un número.
- **Math.cos(x):** Devuelve el coseno de un número.
- **Math.cosh(x):** Devuelve el coseno hiperbólico de un número.
- **Math.exp(x):** Devuelve ex, donde x es el argumento, y e es la constante de Euler (2.718...), la base de los logaritmos naturales.
- **Math.expm1(x):** Las devoluciones restando 1 exp (x)
- **Math.floor(x):** Devuelve el mayor entero menor que o igual a un número.
- **Math.fround(x):** Devuelve la representación flotante de precisión simple más cercana de un número.
- **Math.hypot([x[, y[, …]]]):** Devuelve la raíz cuadrada de la suma de los cuadrados de sus argumentos.
- **Math.imul(x, y):** Devuelve el resultado de una multiplicación de enteros de 32 bits.
- **Math.log(x):** Devuelve el logaritmo natural (log, también ln) de un número.
- **Math.log1p(x):** Devuelve el logaritmo natural de x + 1 (loge, también ln) de un número.
- **Math.log10(x):** Devuelve el logaritmo en base 10 de x.
- **Math.log2(x):** Devuelve el logaritmo en base 2 de x.
- **Math.max([x[, y[, …]]]):** Devuelve el mayor de cero o más números.
- **Math.min([x[, y[, …]]]):** Devuelve el más pequeño de cero o más números.
- **Math.pow(x, y):** Las devoluciones de base a la potencia de exponente, que es, baseexponent.
- **Math.random():** Devuelve un número pseudo-aleatorio entre 0 y 1.
- **Math.round(x):** Devuelve el valor de un número redondeado al número entero más cercano.
- **Math.sign(x):** Devuelve el signo de la x, que indica si x es positivo, negativo o cero.
- **Math.sin(x):** Devuelve el seno de un número.
- **Math.sinh(x):** Devuelve el seno hiperbólico de un número.
- **Math.sqrt(x):** Devuelve la raíz cuadrada positiva de un número.
- **Math.tan(x):** Devuelve la tangente de un número.
- **Math.tanh(x):** Devuelve la tangente hiperbólica de un número.
- **Math.toSource():** Devuelve la cadena "Matemáticas".
- **Math.trunc(x):** Devuelve la parte entera del número x, la eliminación de los dígitos fraccionarios.

2.4.- OBJETO GLOBAL

Cada vez que se ejecuta el intérprete de Javascript se crea un objeto global que contiene las propiedades y métodos que podemos llamar como si fueran constantes y funciones. En estos casos no es necesario incluir el nombre del objeto y un punto para llamarlas.

Este objeto global no es accesible como el resto de objeto de Javascript y solo podemos utilizar sus propiedades y métodos como constantes y funciones globales.

2.4.1.- INFINITY

Constante especial que indica un número que es mayor que el mayor número de coma flotante. El infinito negativo (-Infinity) indica que es un número menor que el menor número de coma flotante.

```
1 / 0 === Infinity   // true
```

2.4.2.- NAN

Constante especial que especifica que una expresión no es un número. La constante NaN (no es un número) es un miembro del objeto global y está disponible en todo momento.
Los operadores == y === no pueden utilizarse con NaN por lo que es necesario utilizar la función isNaN.

2.4.3.- UNDEFINED

Constante especial con un valor que nunca ha sido definido. La constante undefined es una propiedad del objeto global. Cuando se ha declarado una variable pero no se ha inicializado, su valor es undefined, pero también podemos asignar este valor a una variable.

```
let sindefinir = undefined; // no es necesario, pero es perfectamente válido
```

Hay que tener en cuenta que si no se ha declarado una variable no se puede comparar con la constante undefined ya que la variable como tal no existe y se debería utilizar typeof para comparar su resultado con la cadena "undefined".

```
let declarada;
if (declarada === undefined) {
    console.log('sin inicializar');
}
```

2.4.4.- DECODEURI()

Devuelve un identificador uniforme de recursos (URI) previamente codificado. Se debe usar la función decodeURI en lugar de la función en desuso unescape.

`decodeURI(cadenaURI)`: `cadenaURI` es el valor que representa un URI codificado.

La función decodeURI devuelve una cadena con el URI sin codificar o lanza un error URIError si el parámetro cadenaURI no es válido. Normalmente la codificación del URI se realizará con la función encodeURI.

```
let URIcodificada='https://www.todojs.com/%20m%C3%A1s%20informaci%C3%B3n';
console.log(decodeURI(URIcodificada));  // https://www.todojs.com/ más información
```

2.4.5.- DECODEURICOMPONENT()

Devuelve un componente válido de un identificador uniforme de recursos (URI) previamente codificado.

```
decodeURIComponent(cadenaURI)
```

cadenaURI es el valor que representa un componente de URI que queremos descodificar.

La función `decodeURIComponent` devuelve un URI codificado, normalmente por medio de la función `encodeURIComponent`.

```
let URIcodificada = 'http%3A%2F%2Fwww.todojs.com%2F';
```

```
console.log(decodeURIComponent(URIcodificada));//https://www.todojs.co
m/
```

2.4.6.- ENCODEURI()

Codifica un identificador uniforme de recursos (URI). Se debe usar la función encodeURI en lugar de la función en desuso, escape.

```
encodeURI(cadenaURI)
```
`cadenaURI` es la URI que deseamos codificar.

La función encodeURI devuelve una cadena con el URI codificado o lanza un error **URIError** si el parámetro cadenaURI no es válido. La cadena devuelta se puede decodificar con la función decodeURI.

```
let URIsincodificar = 'https://www.todojs.com/ más información';
console.log(encodeURI(URIsincodificar));//https://www.todojs.com/%20m%
C3%A1s%20informaci%C3%B3n
```

La función encodeURI no codifica los caracteres **:, /, ;** y **?**. Se puede usar la función encodeURIComponent para obtener una URI donde se codifiquen estos caracteres.

2.4.7.- ENCODEURICOMPONENT()

Codifica una cadena de texto como un componente válido de un identificador uniforme de recursos (URI), incluidos todos los caracteres especiales.

```
encodeURIComponent(cadenaURI)
```

`cadenaURI` es el valor que representa un componente de URI que queremos codificar.
La función encodeURIComponent devuelve un URI codificado y su resultado es decodificado por la función decodeURIComponent que devuelve la cadena original. La función encodeURIComponent codifica todos los caracteres, excepto las letras, dígitos, y los siguientes caracteres especiales: **-, _, ., !, ~, *, ', (y)**.

```
var URIsincodificar = 'https://www.todojs.com/';
console.log(encodeURIComponent(URIsincodificar));//http%3A%2F%2Fwww.to
dojs.com%2F
```

¿Por qué necesito la codificación URL?

La especificación RFC 1738 URL especifica que sólo un pequeño conjunto de caracteres se puede utilizar en un URL. Los caracteres son:

- A to Z (ABCDEFGHIJKLMNOPQRSTUVWXYZ)
- a to z (abcdefghijklmnopqrstuvwxyz)
- 0 to 9 (0123456789)
- $ (Dollar Sign)
- - (Hyphen / Dash)
- _ (Underscore)
- . (Period)

- + (Plus sign)
- ! (Exclamation / Bang)
- * (Asterisk / Star)
- ' (Single Quote)
- ((Open Bracket)
-) (Closing Bracket)

¿Cómo se codifica una URL?

Todos los caracteres problemáticos son reemplazados por un % y un valor de dos dígitos hexadecimal que representa el carácter del juego de caracteres apropiado ISO. Aquí hay unos ejemplos:

- $ (Dollar Sign) becomes %24
- & (Ampersand) becomes %26
- + (Plus) becomes %2B
- , (Comma) becomes %2C
- : (Colon) becomes %3A
- ; (Semi-Colon) becomes %3B
- = (Equals) becomes %3D
- ? (Question Mark) becomes %3F
- @ (Commercial A / At) becomes %40

2.4.8.- EVAL()

Evalúa código Javascript de forma dinámica y lo ejecuta.

```
eval(codigo)
```

código es la cadena de caracteres que contiene código válido de Javascript que se evaluará y ejecutará.
La función eval permite la ejecución dinámica de código fuente de Javascript, que se ejecutará en el mismo contexto que en la que se realiza la llamada a la función eval.

```
let hoy;
eval("hoy = new Date();");
console.log(hoy.toISOString());
```

El uso de eval es muy controvertido y se ha extendido la expresión -en inglés-: "eval is evil" (eval es el diablo).

2.4.9.- ISFINITE()

Determina si un número es un número finito válido.

```
isFinite(numero)
```

numero, es cualquier valor que queramos comprobar si es un número finito válido. Si el valor no es numérico Javascript convertirá el valor a ese tipo si es posible.

La función isFinite devuelve true si el numero es cualquier valor distinto de NaN, infinito negativo o infinito positivo, en cuyo caso devuelve false.

```
isFinite("hola");                  // false
isFinite(Number.MAX_VALUE * 10);   // false
isFinite(1000);                    // true
isFinite("1000");                  // true
```

2.4.10.- ISNAN()

Devuelve un valor lógico que indica si un valor es `NaN` (no es un número).

```
isNaN(valor)
```

valor, es el valor que se quiere comprobar si es un número o puede ser convertido a un número.
isNan devuelve true si al intentar convertir el valor al tipo número se obtiene NaN (no es un número) o false si es un número válido.

```
isNaN("ABC");  // true

isNaN("100");  // false
isNaN(100);    // false
```

2.4.11.- PARSEFLOAT()

Convierte una cadena en un número de punto flotante.

```
parseInt(cadenaNum)
```

2.4.12.- PARSEINT()

Convierte una cadena en un entero.

```
parseInt(cadenaNum, [radix])
```

`cadenaNum`, es la cadena que se va a convertir en número.
`[radix]`, opcional, es el valor entre 2 y 36 que define la base del número que se incluye en cadenaNum. Si no se pasa este argumento las cadenas con un prefijo 0x se consideran hexadecimales, todas las demás se consideran decimales.

```
parseInt("abc");      // NaN
parseInt("12abc");    // 12
parseInt('0xA', 16); // 10
```

Se puede comprobar si ha devuelto NaN mediante la función isNaN.

2.5.- OBJETO NUMBER

Este objeto es el wrapper correspondiente al tipo de datos primitivo empleado para los números. Su uso no es muy habitual pero podemos utilizarlo si necesitamos acceder a algunas características de los números en JS, como por ejemplo el mayor número que podemos almacenar

2.5.1.- PROPIEDADES DE NUMBER

- **Number.MAX_VALUE:** El mayor número que se puede representar en JavaScript.
- **Number.MIN_VALUE:** Número más cercano a cero que se puede representar en JavaScript.
- **Number.NEGATIVE_INFINITY:** Un valor que es menor que el mayor número negativo que se puede representar en JavaScript.
- **Number.NaN:** Un valor que no es un número.
- **Number.POSITIVE_INFINITY:** Un valor mayor que el mayor número que se puede representar en JavaScript.

2.5.2.- MÉTODOS DE NUMBER

- **toFixed(n):** Devuelve un String con el número o variable sobre el que se invoca el con tantos decimales como indique el parámetro n. Si n es cero o vacío, redondea al entero más próximo. Si es x.50 devuelve el entero inmediato superior si x es positivo o inferior si x es negativo. No genera notación exponencial.

```
let num = 3.1416;
alert('num vale ' + num.toFixed(2));
// num vale "3.14 "
```

- **isNaN(x):** Evalúa a true si x es un valor NaN o a false en caso contrario.

```
alert('¿0/0 vale NaN?: ' + isNaN(0/0)); // true
```

- **toPrecision(n):** Devuelve un String con el número o variable sobre el que se invoca el con un número de dígitos significativos especificado por el número de decimales n. Si n es vacío devuelve el número sin modificar. No admite n cero. Si el número es demasiado grande para representar su parte entera, genera notación exponencial.

```
let num = (3.1416).toPrecision(2);
// num vale 3.1, hay dos dígitos significativos.
```

- **valueOf(x):** Este devuelve la representación como tipo primitivo de un objeto. En el caso de un Number, hace lo mismo que el toString().

```
alert (A1.valueOf());
//Mismo resultado que alert(A1);
```

- **toString():** Devuelve la representación como String de un objeto Number.

```
alert (A1.toString());
//Mismo resultado que alert(A1);
```

Con el objeto Number hay que tener en cuenta que algunas de sus propiedades se usan sin instanciación de ningún tipo. Es decir, si quiero averiguar el mayor número que podemos almacenar no instanciaremos ningún objeto.

2.6.- OBJETO STRING

Este objeto es el wrapper para trabajar con cadenas. Habitualmente no lo utilizamos sino que empleamos directamente sus métodos con una variable de tipo cadena.

2.6.1.- PROPIEDADES DE STRING

- **Length:** La clase String sólo tiene una propiedad: length, que guarda el número de caracteres del String.

2.6.2.- MÉTODOS DE STRING

- **charAt(indice):** Devuelve el carácter que hay en la posición indicada como índice. Las posiciones de un string empiezan en 0.
- **charCodeAt(indice)**: devuelve un número indicando el valor Unicode del carácter en el índice proporcionado.
```
"ABC".charCodeAt(0) // returns 65, el código Unicode de A
```
- **concat(cadena2,cadena3, ...,cadenaN):** combina el texto de dos o más cadenas y devuelve una nueva cadena.
- **fromCharCode(num1, ..., numN):** devuelve una cadena creada mediante el uso de una secuencia de valores Unicode especificada.
- **indexOf(carácter,desde):** Devuelve la posición de la primera vez que aparece el carácter indicado por parámetro en un string. Si no encuentra el carácter en el string devuelve -1. El segundo parámetro es opcional y sirve para indicar a partir de que posición se desea que empiece la búsqueda.
- **lastIndexOf(carácter,desde):** Busca la posición de un carácter exáctamente igual a como lo hace la función indexOf pero desde el final en lugar del principio. El segundo parámetro indica el número de caracteres desde donde se busca, igual que en indexOf.
- **match(regexp)**: se usa para obtener todas las ocurrencias de una expresión regular dentro de una cadena.
```
let str = "The rain in SPAIN stays mainly in the plain";
let res = str.match(/ain/g); // ain,ain,ain
```
- **replace(substring_a_buscar,nuevoStr):** Implementado en Javascript 1.2, sirve para reemplazar porciones del texto de un string por otro texto, por ejemplo, podríamos uilizarlo para reemplazar todas las apariciones del substring "xxx" por "yyy". El no reemplaza en el string, sino que devuelve un resultante de hacer ese reemplazo. Acepta expresiones regulares como substring a buscar.
- **search(expresionregular)**: ejecuta la búsqueda que encaje entre una expresión regular y el objeto String desde el que se lo llama.
- **slice(inicioTrozo[, finTrozo])**: extrae una sección de una cadena y devuelve una cadena nueva. slice extrae hasta, pero sin incluir finalTrozo. string.slice(1,4) extrae del segundo carácter hasta el cuarto carácter (caracteres con índice 1, 2 y 3).
- **split(separador):** Este sólo es compatible con javascript 1.1 en adelante. Sirve para crear un vector a partir de un String en el que cada elemento es la parte del String que está separada por el separador indicado por parámetro.
- **substr(inicio[, longitud])**: devuelve los caracteres de una cadena que comienzan en una localización especificada y de acuerdo al número de caracteres que se especifiquen.

- **substring(inicio,fin):** Devuelve el substring que empieza en el carácter de inicio y termina en el carácter de fin. Si intercambiamos los parámetros de inicio y fin también funciona. Simplemente nos da el substring que hay entre el carácter menor y el mayor.
- **toLowerCase():** Pone todas los caracteres de un string en minúsculas.
- **toUpperCase():** Pone todas los caracteres de un string en mayúsculas.
- **toString():** Este lo tienen todos los objetos y se usa para convertirlos en cadenas.
- **valueOf()**:opuesto a toString(). En este caso si la cadena contiene un número se extrae su valor primitivo.

  Hasta aquí hemos visto los métodos que nos ayudarán a tratar cadenas. Ahora vamos a ver otros métodos que son menos útiles, pero hay que indicarlos para que quede constancia de ellos. Todos sirven para aplicar estilos a un texto y es como si utilizásemos etiquetas HTML. Veamos cómo.

  - **anchor(name):** Convierte en un ancla (sitio a donde dirigir un enlace) una cadena de caracteres usando como el atributo name de la etiqueta <A> lo que recibe por parámetro.
  - **big():** Aumenta el tamaño de letra del string. Es como si colocásemos en un string al principio la etiqueta <BIG> y al final </BIG>.
  - **bold():** Como si utilizásemos la etiqueta <B>.

```
document.write(cadena.bold); // escribe la cadena en negritas
alert(enlace.anchor("BIO")); // genera <a name="BIO">Biografía</a>
```

  - **fixed():** Para utilizar una fuente monoespaciada, etiqueta <TT>.
  - **fontColor(color):** Pone la fuente a ese color. Como utilizar la etiqueta <FONT color=el_color_indicado>.
  - **fontSize(tamaño):** Pone la fuente al tamaño indicado. Como si utilizásemos la etiqueta <FONT> con el atributo size.
  - **italics():** Pone la fuente en cursiva. Etiqueta <I>.
  - **link(url):** Pone el texto como un enlace a la URL indicada. Es como si utilizásemos la etiqueta <A> con el atributo href indicado como parámetro.
  - **small():** Es como utilizar la etiqueta <SMALL>
  - **strike():** Como utilizar la etiqueta <STRIKE>, que sirve para que el texto aparezca tachado.
  - **sub():** Actualiza el texto como si se estuviera utilizando la etiqueta <SUB>, de subíndice.
  - **sup():** Como si utilizásemos la etiqueta <SUP>, de superíndice

2.7.- OBJETO DATE.

Este objeto nos permite trabajar con fechas. Puede almacenar la fecha con gran precisión, de ahí que lo usaremos en la práctica del juego de la Serpiente. Entre sus muchos métodos algunos serán de sólo lectura, otros de escritura y otros para convertir entre diferentes formatos de fecha.
Un objeto Date contiene un número que representa un instante de tiempo concreto en milisegundos. Si el valor de un argumento es mayor que su intervalo o es un número negativo, los demás valores almacenados

se modifican consecuentemente. Por ejemplo, si especifica 150 segundos, JavaScript vuelve a definir ese número como dos minutos y 30 segundos.
Si el número es NaN, el objeto no representa un instante específico de tiempo. Si no pasa ningún parámetro al objeto Date, se inicializa con la hora actual (UTC). Se debe asignar un valor al objeto antes de poder usarlo.
El intervalo de fechas que se puede representar en un objeto Date abarca el período comprendido, aproximadamente, entre los 285.616 años antes y después del 1 de enero de 1970.

Funciones de la clase:

- **Date.now()**: Número de milisegundos que hay entre la medianoche del 1 de enero de 1970 y la fecha y hora actuales.
- **Date.parse(dateVal)**: Analiza una cadena que contiene una fecha y devuelve el número de milisegundos transcurridos entre esa fecha y la medianoche del 1 de enero de 1970.
- **Date.UTC(year, month, day[, hours[, minutes[, seconds[,ms]]]])**: Devuelve el número de milisegundos transcurrido entre la medianoche del 1 de enero de 1970 según el horario universal coordinado (UTC) (o GMT) y la fecha especificada.

2.7.1.- MÉTODOS DE DATE

- **getDate**: Devuelve el valor de día del mes usando la hora local.
- **getDay**: Devuelve el valor de día de la semana usando la hora local.
- **getFullYear**: Devuelve el valor de año usando la hora local.
- **getHours**: Devuelve el valor de horas usando la hora local.
- **getMilliseconds**: Devuelve el valor de milisegundos usando la hora local.
- **getMinutes**: Devuelve el valor de minutos usando la hora local.
- **getMonth**: Devuelve el valor de mes usando la hora local.
- **getSeconds**: Devuelve el valor de segundos usando la hora local.
- **getTime**: Devuelve el valor de tiempo en un objeto Date en milisegundos desde la medianoche del 1 de enero de 1970.
- **getTimezoneOffset**: Devuelve la diferencia en minutos entre la hora del equipo host y la hora universal coordinada (UTC).
- **getUTCDate**: Devuelve el valor de día del mes usando la hora UTC.
- **getUTCDay**: Devuelve el valor de día de la semana usando la hora UTC.
- **getUTCFullYear**: Devuelve el valor de año usando la hora UTC.
- **getUTCHours**: Devuelve el valor de horas usando la hora UTC.
- **getUTCMilliseconds**: Devuelve el valor de milisegundos usando la hora UTC.
- **getUTCMinutes**: Devuelve el valor de minutos usando la hora UTC.
- **getUTCMonth**: Devuelve el valor de mes usando la hora UTC.
- **getUTCSeconds**: Devuelve el valor de segundos usando la hora UTC.
- **getVarDate**: Devuelve el valor VT_DATE de un objeto Date.
- **getYear**: Devuelve el valor de año.
- **hasOwnProperty**: Devuelve un valor booleano que indica si un objeto tiene una propiedad con el nombre especificado.

- **isPrototypeOf**: Devuelve un valor booleano que indica si un objeto existe en la cadena de prototipo de otro objeto.
- **propertyIsEnumerable**: Devuelve un valor booleano que indica si una propiedad especificada forma parte de un objeto y si se puede enumerar.
- **setDate**: Establece el día del mes numérico usando la hora local.
- **setFullYear**: Establece el valor de año usando la hora local.
- **setHours**: Establece el valor de horas usando la hora local.
- **setMilliseconds**: Establece el valor de milisegundos usando la hora local.
- **setMinutes**: Establece el valor de minutos usando la hora local.
- **setMonth**: Establece el valor de mes usando la hora local.
- **setSeconds**: Establece el valor de segundos usando la hora local.
- **setTime**: Establece el valor de fecha y hora en el objeto Date.
- **setUTCDate**: Establece el día numérico del mes usando la hora UTC.
- **setUTCFullYear**: Establece el valor de año usando la hora UTC.
- **setUTCHours**: Establece el valor de horas usando la hora UTC.
- **setUTCMilliseconds**: Establece el valor de milisegundos usando la hora UTC.
- **setUTCMinutes**: Establece el valor de minutos usando la hora UTC.
- **setUTCMonth**: Establece el valor de mes usando la hora UTC.
- **setUTCSeconds**: Establece el valor de segundos usando la hora UTC.
- **setYear**: Establece el valor de año usando la hora local.
- **toDateString**: Devuelve una fecha como un valor de cadena.
- **toGMTString**: Devuelve una fecha convertida en cadena utilizando la hora media de Greenwich (GMT).
- **toISOString**: Devuelve una fecha como un valor alfanumérico en formato ISO.
- **toJSON**: Se utiliza para transformar datos de un tipo de objeto antes de la serialización JSON.
- **toLocaleDateString**: Devuelve una fecha como un valor de cadena apropiado para la configuración regional actual del entorno host.
- **toLocaleString**: Devuelve un objeto convertido en cadena usando la configuración regional actual.
- **toLocaleTimeString**: Devuelve una hora como un valor de cadena apropiado para la configuración regional actual del entorno host.
- **toString**: Devuelve una representación en forma de cadena de un objeto.
- **toTimeString**: Devuelve una hora como un valor de cadena.
- **toUTCString**: Devuelve una fecha convertida en cadena usando la hora UTC.
- **valueOf**: Devuelve el valor primitivo del objeto especificado.

2.8.- OBJETO ARRAY.

Para crear un vector con datos podemos utilizar la forma básica que consiste en declarar los valores entre corchetes:

```
let dias = ["Domingo", "Lunes", "Martes", "Miércoles", "Jueves",
"Viernes", "Sábado"];
alert(dias[2]); // produce "Martes"
```

Sin embargo, también podríamos hacerlo utilizando el constructor del objeto Array, que es la clase wrapper para los vectores:

```
let dias = new Array("Domingo", "Lunes", "Martes", "Miércoles",
"Jueves", "Viernes", "Sábado");
alert(dias[2]); // produce "Martes"
```

Como ves si utilizamos el constructor los valores se encierran entre paréntesis y separados por comas. Sin embargo si en el momento de crear el array no conocemos su contenido podemos crearlo vacío y luego agregar los elementos separadamente:

```
let dias = new Array(); // un array sin elementos
dias[0] = "Domingo";
dias[1] = "Lunes";
let mivector = new Array(10); // aquí creamos un vector de 10
elementos, que estarán en las posiciones de la 0 a la 9.
Mivector[5] = 35.68; // almacenamos un 35.68 en la posición 5
```

También podríamos utilizar los métodos de la clase Array para agregar elementos. El siguiente código es equivalente al anterior:

```
let dias = new Array(); // un array sin elementos
días.push("Domingo"); // push agrega un elemento al final
días.push("Lunes");
```

Para acceder a un elemento de un array se utilizan los corchetes.

```
let vector = new Array(); // un array sin elementos
vector[0] = true; vector[1] = 3; vector[4] = new Date(); vector[5]
=new Image();
```

Con la flexibilidad del lenguaje dentro de los elementos de un vector podrá haber variables, funciones, otros vectores, etc. Observa este código:

```
let vector = new Array(); // un array sin elementos
function multiplicar(num1, num2) {
   return (num1 * num2);
}
vector[0] = function(a, b) {return a+b};
vector[1] = multiplicar;
vector[2] = new Array(4, 14, 24, 34);
alert(vector[0](5,15)); // ejecuta la suma de 5 y 15
alert(vector[1](4,2)); // ejecuta la multiplicación de 4 y 2
alert(vector[2][1]); // produce 14
```

Como ves en el código anterior el primer elemento del vector contiene una función anónima. El segundo elemento contiene también una función llamada multiplicar. El tercer elemento contiene otro vector (técnica empleada para crear matrices).

2.8.1.- PROPIEDADES DE ARRAY

- **arrayObj.length**: Obtiene o establece la longitud de la matriz

2.8.2.- MÉTODOS DE ARRAY

- **concat (matriz):** Devuelve una matriz nueva que se compone de una combinación de dos matrices.
- **entries:** Devuelve un iterador que contiene los pares clave-valor de la matriz.
- **every:** Comprueba si una función de devolución de llamada definida devuelve true para todos los elementos de una matriz.
- **fill:** Rellena una matriz con un valor especificado.
- **filter:** Llama a una función de devolución de llamada definida en cada elemento de una matriz y devuelve una matriz de valores para los que la función de devolución de llamada devuelve true.
- **findIndex:** Devuelve un valor de índice del primer elemento de matriz que cumple los criterios de prueba especificados en una función de devolución de llamada.
- **forEach:** Llama a una función de devolución de llamada definida para cada elemento de una matriz.
- **hasOwnProperty:** Devuelve un valor booleano que indica si un objeto tiene una propiedad con el nombre especificado.
- **IndexOf (matriz):** Devuelve el índice de la primera aparición de un valor de una matriz.
- **isPrototypeOf:** Devuelve un valor booleano que indica si un objeto existe en la cadena de prototipo de otro objeto.
- **join:** Devuelve un objeto String formado por todos los elementos de una matriz concatenados.
- **keys:** Devuelve un iterador que contiene los valores de índice de la matriz.
- **lastIndexOf (matriz):** Devuelve el índice de la última aparición de un valor especificado de una matriz.
- **map:** Llama a una función de devolución de llamada definida en cada elemento de una matriz y devuelve una matriz que contiene los resultados.
- **pop:** Quita el último elemento de una matriz y lo devuelve.
- **propertyIsEnumerable:** Devuelve un valor booleano que indica si una propiedad especificada forma parte de un objeto y si es enumerable.
- **push:** Anexa elementos nuevos a una matriz y devuelve la nueva longitud de la matriz.
- **reduce**: Acumula un solo resultado llamando a una función de devolución de llamada definida para todos los elementos de una matriz. El valor devuelto de la función de devolución de llamada es el resultado acumulado y se proporciona como argumento en la llamada siguiente a la función de devolución de llamada.
- **reduceRight:** Acumula un solo resultado llamando a una función de devolución de llamada definida para todos los elementos de una matriz en orden descendente. El valor devuelto de la función de devolución de llamada es el resultado acumulado y

se proporciona como argumento en la llamada siguiente a la función de devolución de llamada.

- **Reverse**: Devuelve un objeto Array con los elementos invertidos.
- **shift**: Quita el primer elemento de una matriz y lo devuelve.
- **slice (matriz):**Devuelve una sección de una matriz.
- **some**: Comprueba si una función de devolución de llamada definida devuelve true para cualquier elemento de una matriz.
- **sort**: Devuelve un objeto Array con los elementos ordenados.
- **splice**: Quita los elementos de una matriz y, si es necesario, inserta en su lugar elementos nuevos y devuelve los elementos eliminados.
- **toLocaleString**: Devuelve una cadena usando la configuración regional actual.
- **toString**: Devuelve una representación de cadena de una matriz.
- **unshift**: Inserta elementos nuevos al principio de una matriz.
- **valueOf**: Obtiene una referencia a la matriz.
- **values**: Devuelve un iterador que contiene los valores de la matriz.

2.8.3.- CREACIÓN DE ARRAY

Por ejemplo si tenemos las siguientes variables:

```
let coches = [ 'Seat', 'Audi', 'BMW', 'Toyota' ];
let numeros = [ 1, 5, 3, 9, 6, 4 ];
let diferentes = [ 'Pepe', 5, 'Juan', false ];
```

También es posible utilizar la sintaxis `new Array()` y luego ir asignado valores a los diferentes elementos, como por ejemplo:

```
let coches = new Array();
coches[0] = 'Seat';
coches[1] = 'Audi';
coches[2] = 'BMW';
coches[3] = 'Toyota';
```

En JavaScript también es posible utilizar arrays asociativos, es decir, asociaciones clave-valor. Veamos un ejemplo:

```
let edades = new Array();
edades['Juan'] = 20;
edades['Ana'] = 18;
edades['Pedro'] = 25;
console.log(edades);
console.log(edades['Juan']);
console.log(edades['Ana']);
console.log(edades['Pedro']);
```

2.8.4.- RECORRER UN ARRAY

Para recorrer un array podemos utilizar uno de los bucles que ya conocemos. Por ejemplo, dado el siguiente array:

```
let numeros = [ 1, 5, 3, 9, 6, 4 ];
```

Podríamos utilizar un bucle `for` para recorrerlo:

```
for (let i = 0; i < numeros.length; i++) {
    console.log(numeros[i]);
}
```

También existe un bucle especial para recorrer arrays en JavaScript; el denominado `for-in`:

```
for (let i in numeros) {
    console.log(numeros[i]);
}
```

Incluso podemos utilizar otras construcciones como la función `forEach`:

```
numeros.forEach(numero => console.log(numero));
```

Es más, podemos utilizar las funciones `map`, `reduce` y `filter` para realizar algunas acciones comunes con los arrays:

```
let cuadrados = numeros.map(numero => {return numero * numero});
console.log(cuadrados);
```

En el siguiente enlace se muestran ejemplos detallados de cómo utilizar estas nuevas características.

2.8.5.- ACCEDER A LOS OBJETOS DEL ARRAY

Una vez creado un array, siempre podremos añadir nuevos elementos de forma dinámica utilizando un indice no utilizado.

```
numeros[6] = 15;
console.log(numeros);
```

Debemos tener en cuenta que si no añadimos el nuevo elemento al final, las posiciones intermedias se quedarán con valores indefinidos.

Podemos eliminar un elemento del array mediante la función `delete`, pero esta función no reduce el tamaño del array, simplemente borra el elemento:

```
delete numeros[6];
console.log(numeros.length);     // Muestra por consola: 7
console.log(numeros[6]);         // Muestra por consola: undefined
```

Si lo que queremos es también reducir su tamaño deberemos utilizar la función `splice`; a esta función le indicamos la posición a partir de la cual empezamos a eliminar y el número de elementos:

```
numeros.splice(5, 2);
console.log(numeros);        // Muestra por consola: [ 1, 5, 3, 9, 6 ]
```

2.8.6.- ARRAY MULTIDIMENSIONALES

Como ya sabemos, un array puede almacenar elementos de diferentes tipos, por lo que un elemento de un array, a su vez puede ser otro array y a eso es a lo que se conoce como arrays multidimensionales.
Veamos el mismo ejemplo anterior, pero con un array multidimensional.

```
let personajes = [
    [ 'Bob Esponja', 'amarillo'],
    [ 'Calamardo', 'beige' ],
    [ 'Patricio', 'rosa' ]
];

for (let i in personajes) {
    console.log(`${personajes[i][0]} es de color {personajes[i][1]}`);
}
```

Revisa esta página: Uso de las funciones `map`, `filter` y `reduce`.

2.9.- UTILIZANDO INTERVALOS DE TIEMPO.

JS ofrece dos funciones que nos permitirán ejecutar un código cada cierto intervalo de tiempo. Estas funciones son interesantes porque son en sí mismas eventos que se disparan automáticamente en el tiempo. Ambas funciones son métodos del objeto window pero las veremos aquí para hacer la siguiente práctica:

setInterval(f, x): Esta función lanzará la ejecución de la función f cada x milisegundos.
Si queremos finalizar un setInterval tendremos que asociarlo a una variable. Observa el código:

```
let mivble = setInterval(calcular, 1000);//dispara calcular cada segundo
clearInterval(mivble);
```

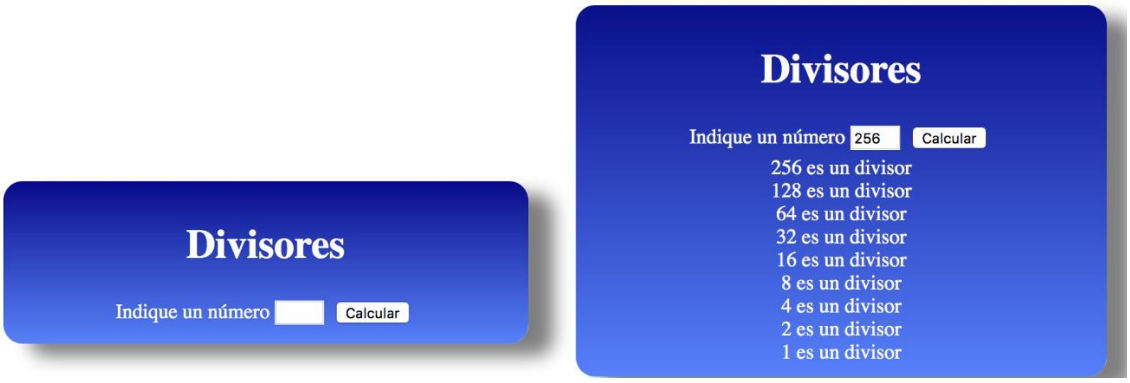
setTimeout(f, x): Esta función es muy parecida a la anterior pero en este caso se esperará x milisegundos para lanzar la ejecución de la función f una única vez. Por lo demás es exactamente igual a setInterval. Se detiene de la misma manera pero utilizando clearTimeout(variable)

ACTIVIDADES

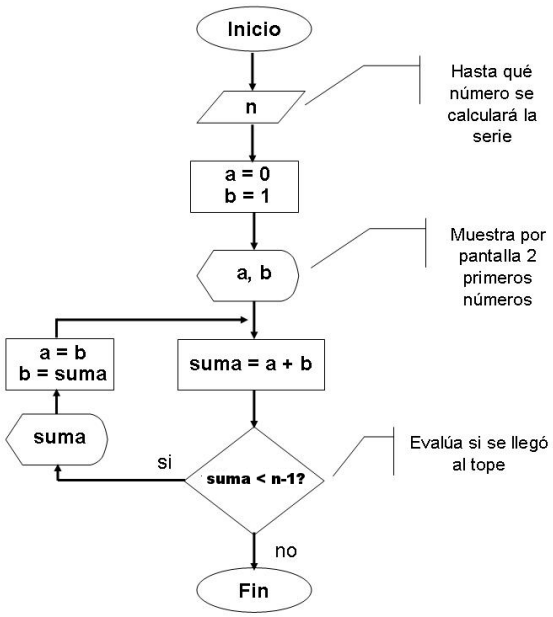
Utiliza JavaScript para resolver las siguientes actividades

1. Calcula los divisores de un número. Si el dato que se indica no es un número se mostrará un error. Utiliza un en la página para incluir la información, al principio no tiene código. Lo que haremos será insertarle código con la propiedad **innerHTML**. Esta propiedad nos permite insertar todo el código HTML que queramos dentro de un elemento. Ejemplo:

```
Id_div.innerHTML = "<img src='logo.jpg'/>"; // insertamos una imagen
```

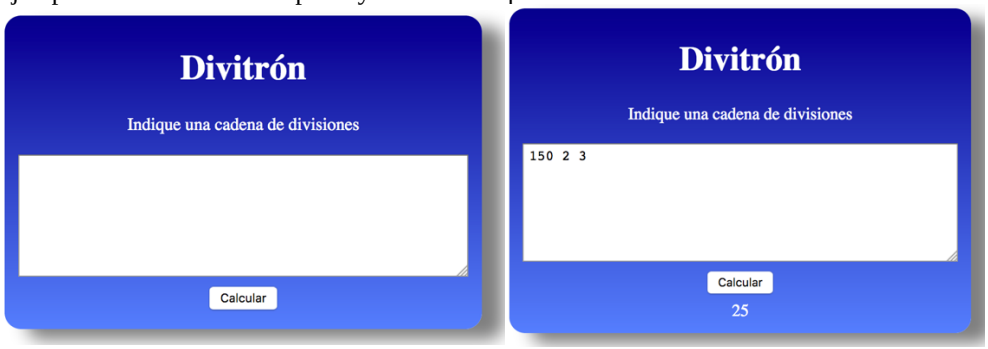


2. Calcula sucesión de Fibonacci, utiliza el siguiente algoritmo (no usar recursividad).



3. Calcula sucesión de Fibonacci utilizando una función recursiva.
4. Define una calculadora que realice las operaciones indicadas

5. Añada 4 botones para realizar la raíz cuadrada de un número, generar un número aleatorio entre 0 y 100, exponenciales de dos números y arcocoseno (recuerde que éste último debe dar el resultado en grados, arccos(0)=90º).
6. Defina la siguiente página donde se realizarán múltiples divisiones. Utilice un **textarea**. Para el ejemplo 150 se ha dividido por 2 y el resultado por 3.



Utilice `<span id="resultado_division"></span>` para mostrar el resultado de la división. Muestre sólo 2 números decimales, en el caso de resultados reales y ninguno cuando el resultado sea entero.
¿Qué ocurrirá en su código cuando ponga un valor incorrecto? Por ejemplo texto.
Mejore su código para que cuando alguien utilice comas en un número, por ejemplo 3,5 la aplicación lo interprete como un número real 3.5

7. [**Opcional**]: Utilizar recursividad para resolver el ejercicio anterior.
8. Muestra una página donde al cargar la página se vea una fecha. La fecha tiene que tener el siguiente formato:

9. Modifique el anterior ejercicio para que se muestren las horas minutos y segundos.

Utilice `setInterval` para que cambie la fecha y hora en "tiempo real" (cada segundo). La siguiente captura se ha realizado tiempo después de la anterior

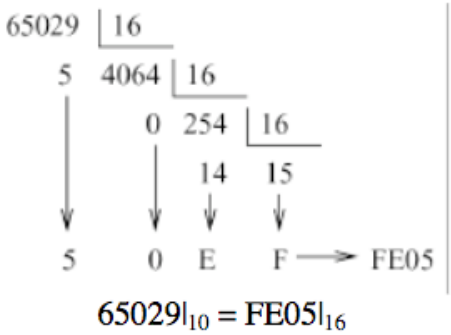
10. Defina una aplicación donde al pasar una fecha, por ejemplo la de su nacimiento, devuelva el número de años, meses, días y horas transcurridas desde esa fecha. También indique el número de años bisiestos transcurridos. Para saber cómo se sabe si un año es bisiesto visita el apartado "Algoritmo computacional" en https://es.wikipedia.org/wiki/A%C3%B1o_bisiesto .
11. Defina una función recursiva que convierta un número decimal a binario.

12. Construya una función que convierta un número decimal en una cadena que represente el valor del número en hexadecimal (base 16). A continuación, generalice la función para convertir un número decimal en un número en base B (con B<10).

Recordatorio: El cambio de base se realiza mediante divisiones sucesivas por 16 en las cuales los restos determinan los dígitos hexadecimales del número según la siguiente correspondencia:

Resto	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Dígito	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Por ejemplo:



$65029|_{10} = FE05|_{16}$

13. **Salida de un laberinto**: Se trata de encontrar un camino que nos permita salir de un laberinto definido en una matriz NxN. Para movernos por el laberinto, sólo podemos pasar de una casilla a otra que sea adyacente a la primera y no esté marcada como una casilla prohibida (esto es, las casillas prohibidas determinan las paredes que forman el laberinto).

Algoritmo:

- Se comienza en la casilla (0,0) y se termina en la casilla (N-1,N-1)
- Nos movemos a una celda adyacente si esto es posible.
- Cuando llegamos a una situación en la que no podemos realizar ningún movimiento que nos lleve a una celda que no hayamos visitado ya, retrocedemos sobre nuestros pasos y buscamos un camino alternativo.

FUENTES

http://www.maestrosdelweb.com/que-es-javascript/
http://librosweb.es/libro/javascript/
http://librosweb.es/
http://www.aprenderaprogramar.es/index.php?option=com_content&view=article&id=788:tipos-de-datos-javascript-tipos-primitivos-y-objeto-significado-de-undefined-null-nan-ejemplos-cu01112e-&catid=78:tutorial-basico-programador-web-javascript-desde-&Itemid=206
https://www.todojs.com/ref/javascript/global/
https://linkeandoenlared.blogspot.com.es/2012/11/codificar-y-descodificar-url.html