APUNTES 3.6. FUNCIONES EN JAVASCRIPT.

“Las funciones en JavaScript son objetos”.

3.6. Funciones en JavaScript.

Como se ha dicho anteriormente, las funciones en JavaScript son objetos. La función suma, por ejemplo, se puede declarar de la siguiente forma:

```
function suma (a, b){
    return a + b;
}
```

A diferencia de otros lenguajes de programación, esta función se puede crear mediante un constructor llamado Function():

```
var suma = new Function("a", "b", "return a + b");
```

3.6. Funciones en JavaScript.

El resultado de la siguiente línea de código será idéntico tanto si se declara la función de forma clásica como si se crea mediante un constructor:

```
var c = suma (2,2);
```

También se puede declarar la función después de utilizarla. No es lo más común en los lenguajes de programación, pero JavaScript permite crear el siguiente código:

```
var c = suma(2,2);
var suma = new Function("a", "b", "return a + b");
```

3.6. Funciones en JavaScript.

La función flecha o anónima es muy utilizada por los programadores en la actualidad. Sustituye a la expresión de función tradicional, como se puede observar a continuación:

```
function resta (a,b){return a-b;}; //función tradicional JavaScript
let resta = (a,b) =>{return a-b;}; //función flecha 1
let resta = (a,b) =>a-b; //función flecha 2
```

Esta función flecha se utiliza mucho en frameworks y otro tipo de funciones para arrays, como map(), filter(), etc.

3.6. Funciones en JavaScript.

Otras tres líneas de código equivalentes. En este caso, para la función cuadrado, la cual, dado un número devuelve el cuadrado del mismo:

```
let cuadrado = (a) => a*a;
let cuadrado = a => a*a;
let cuadrado = (a) => {return a*a;};
```

En los ejemplos anteriores, las funciones siempre tenían parámetros. Un ejemplo de función flecha sin parámetros sería el siguiente:

```
let hola = () =>"hola";
```

3.6.1. La recursividad en JavaScript.

Es una función o algoritmo recursivo que genera la solución realizando llamadas así mismo. La ventaja es la reducción de la complejidad del problema hasta que se llegue a un punto en la cual ya no haga falta utilizar la recursión, puesto que la resolución es algo directo o simple. En JavaScript, es posible utilizar la recursividad para la resolución de problemas:

En el código siguiente se muestra como se resuelve mediante recursividad el problema de calcular el factorial del número 10.

Recordemos que el factorial de un número, por ejemplo el 5, es: 5! = 5 x 4 x 3 x 2 x 1

3.6.1. La recursividad en JavaScript.

```
<!DOCTYPE html>
<html>
    <head><title>Ejemplo de recursividad</title></head>
    <body>
        <script>
            function factorial(num){
                if (num == 0){
                    return 1;
                }else{
                    return (num * factorial(num-1));
                }
            }
            var numero = factorial (10);
            document.write(numero);
        </script>
    </body>
</html>
```

3.6.1. La recursividad en JavaScript - TAREA 3.3

Teniendo en cuenta el código expuesto en este apartado, crea una función recursiva que muestre la sucesión de Fibonacci. Comprueba en un navegador que la función visualiza correctamente la secuencia deseada.

3.6.2. Los parámetros de las funciones.

En JavaScript, es posible llamar a una función con menos parámetros de los previstos. No dará error en ejecución, pero hay que tener previsto en el código esta eventualidad. Véase un ejemplo de esto en la siguiente función:

```
function suma (a, b){
    if (b === undefined){
        return a + a;
    }
   return a + b;
}
//llamada normal
var c = suma (2,2);
//llamada con un parámetro solo
var c = suma (2);
```

3.6.2. Los parámetros de las funciones.

Siempre que hay que utilizar fórmulas u operaciones matemáticas, es necesario utilizar una librería u objeto matemático. En JavaScript, está el objeto Math, el cual tiene numerosos métodos y propiedades que pueden servir para ejecutar muchas de las necesidades que se tengan en una aplicación.

```
//propiedades
var x1 = Math.LN10; //Devuelve el logaritmo natural de 10 (2,302)
var x2 = Math.E; //Devuelve el valor del número e (2,718)
var x3 = Math.PI; //Devuelve el valor del número pi (3,1416)
```

3.6.2. Los parámetros de las funciones.

```
//métodos
var y1 = Math.abs(-5); //Devuelve el valor absoluto del  parámetro
introducido
var y2 = Math.sqrt(49); //Devuelve la raíz cuadrada de 49
var y3 = Math.random();  //Devuelve un número aleatorio entre cero y uno.
```

3.6.3. Funciones y métodos en objetos de tipo array.

Los arrays son uno de los objetos que tienen más métodos (funciones) para su tratamiento. Algunos de los métodos más útiles y comunes de este objeto:

a) *concat()*.
b) *fill()*.
c) *filter()*.

3.6.3. Funciones y métodos en objetos de tipo array.

Ejemplo del uso de *filter()*:

```
//devolver un array con los números naturales  encontrados [13,1,8,2]
var datos= [13, -5, -9, 1, 8, 2];
function natural(dato){
    return dato >= 0;
}
function buscaNumerosNaturales(){
    document.getElementById( "cualquiercampo").innerHTML = datos.filter(-natural);
}
```

3.6.3. Funciones y métodos en objetos de tipo array.

d) *find()*. Devuelve el valor del primer elemento que pase el test suministrado por parámetros

```
//Objetivo: devolver el primer número natural encontrado (13).
var datos = [13, -5, -9, 1, 8, 2];
function natural (dato){
    return dato>=0;
}
function buscaNumerosNaturales(){
    document.getElementById("cualquiercampo").innerHTML = datos.find(natural);
}
```

3.6.3. Funciones y métodos en objetos de tipo array.

e) *forEach()*. Realiza una llamada a una función por cada uno de los elementos de un array:

```
<p id="diasDeLaSemana">Aquí aparecerán los días de la semana:<br></p>
<script>
    textoDias = document.getElementById("diasDeLaSemana");
    var dias = ["lunes", "martes", "miércoles", "jueves", "viernes", "sábado",
"domingo"];
    function myFunction(item, index){
    textoDias.innerHTML = textoDias.innerHTML +"index[" + index + "]: " + item +
"<br>";
    }
</script>
<button onclick="dias.forEach(myFunction)">Dale!</button>
```

3.6.3. Funciones y métodos en objetos de tipo array.

f) *includes()*. Comprueba si en el array está un elemento especificado.

g) *indexOf()*. Busca en el array un elemento y devuelve su posición.

h) *isArray()*. Comprueba si un objeto determinado es un array.

i) *lastIndexOf()*. Devuelve la posición de un elemento de un array. Importante, este método, al contrario de lo que pudiese pensarse, comienza la búsqueda por el final:

```
var dias = ["lunes","jueves","miércoles", "jueves", "viernes", "sábado", "domingo"];
var pos=dias.lastIndexOf("jueves"); //ahora "pos" valdrá 3 porque busca desde el final.
```

3.6.3. Funciones y métodos en objetos de tipo array.

j) pop(). Elimina un elemento de la última posición de un array. Este método devuelve dicho elemento.

k) push(). Añade un elemento nuevo al array en su última posición. Devuelve la longitud del nuevo array. Con los métodos push() y pop(), se puede implementar una pila.

l) shift(). Elimina el primer elemento del array. Este método devuelve dicho elemento. Útil cuando se desea realizar algún proceso con los elementos de un array, pero no se desea conservarlos. También es útil para hacer una cola de elementos en combinación con push().

m) slice(). Selecciona parte de un array.

3.6.3. Funciones y métodos en objetos de tipo array.

m) slice(). Selecciona parte de un array.

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes","Sábado", "Domingo"];
var finde = dias.slice(1,7);
```

n) sort(). Ordena los elementos de un array.
o) splice(). Método muy útil para añadir o eliminar elementos de un array.
p) toString(). Convierte un array en un string.
q) valueOf(). Devuelve los valores de un array. Puede ser útil para hacer copias de un objeto array.

3.6.3. Funciones y métodos en objetos de tipo array.

Ejemplo de *valueOf()*:

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes","Sábado", "Domingo"];
var dias2 = dias.valueOf();
```

3.6.3. *A) La función para arrays map()*.

Cuando necesitamos realizar una operación por cada elemento de un array debemos realizar un bucle entre todos los elementos del array e ir recuperando y ejecutando cierto código por cada elemento. Para evitarnos ésto, tenemos la función map().

Ejemplo función para arrays *map()*

```
<!DOCTYPE html>
<html>
    <body>
        <p id="resultado">Tenemos el array definido de la siguiente manera:
            let numeros = [1, 2, 3, 4, 5, 6, 7]; cuando pulses el botón aquí aparecerán
            los mismos números incrementados en uno</p>
        <button onclick="dale()">ejecuta</button>
        <script>
            let numeros = [1, 2, 3, 4, 5, 6, 7];
            function dale(){
                let obj = document.getElementById("resultado")
                obj.innerHTML = numeros.map(function(valor){return valor+1});
                /*map irá ejecutando la función incluida como parámetro la cual no tiene
                nombre porque no es necesario. Esta función tomará como parámetro valor (el
                valor de esa posición del array) y devolverá dicho valor aumentado en uno.
                La función map devuelve otro array con el resultado de aplicar la función
                anterior a los elementos de todo el array*/
            }
        </script></body></html>
```

3.6.3. *A) La función para arrays map()*.

La función map devuelve otro array con el resultado de aplicar la función pasada como parámetro a los elementos del array. Map es la función SIN (sin bucles y sin tener que crear un array, puesto que lo hace ella sola).

El prototipo de esta función es el siguiente:

```
array.map(function(elemento, indice, el_array), valor_this)
```

3.6.3. *A) La función para arrays map().*

La función map tiene dos parámetros. El primero es la función callback que, a su vez, tiene tres parámetros, y el segundo parámetro es valor_this, el cual es opcional y se podrá utilizar como “this” para referenciar al objeto de la llamada.
En la función callback el único parámetro obligatorio “es elemento” (el primero). “Índice”, que indica el índice del elemento, y el “el_array”, que apunta al array protagonista de la llamada son opcionales.

```
let numeros = [1, 2, 3, 4, 5, 6, 7];
            function dale(){
                let obj = document.getElementById("resultado")
                obj.innerHTML = numeros.map(function(valor,indice){return valor+indice});
```

3.6.3. *A) La función para arrays map().*

El anterior código es una modificación del programa ejemplo primero y su función es sumar a cada elemento del array el índice que ocupa en dicho array.

No hace falta utilizar la función map() obligatoriamente para devolver un array. Puedes utilizarla para realizar una operación por cada elemento del array:

```
let numeros = [1, 2, 3, 4, 5, 6, 7];
numeros.map(function(valor,indice){console.log( "valor: "+valor + " índice: "
+indice)});
```

3.6.3. *A) La función para arrays map().*

Lo más normal es encontrarnos la función map() en combinación con la función flecha => o función anónima.

```
obj.innerHTML = numeros.map( function(valor,indice){ return valor+indice;});
obj.innerHTML = numeros.map((valor,indice) =>{return valor+indice;});
```

La función flecha es muy común su utilización en muchos frameworks como ReactJS o Angular. La función arrow es un forma intuitiva, fácil y rápida de programar dichos callback.

3.6.3. *B) La función para arrays reduce()*.

Esta función permite iterar un array acumulando las operaciones que se vayan haciendo sobre los elementos en otro llamado acumulador:

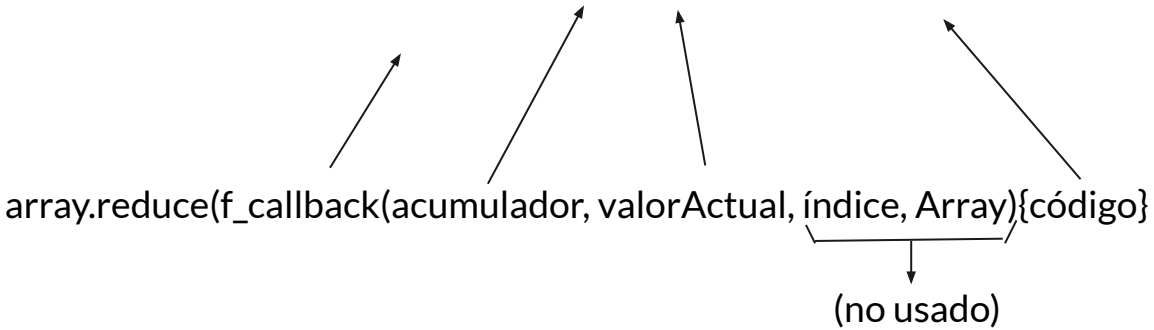
```
let numeros = [1, 2, 3, 4];
console.log (numeros.reduce((acc,v) =>{return acc+v;}));
let numeros2 = ["uno", "dos", "tres", "cuatro"];
console.log (numeros2.reduce((a,b) =>{return a.concat(b);}));
```

En el primero de ellos se sumarán todos los números del array (1+2+3+4), lo cual mostrará por consola en la segunda línea el valor 10.
El segundo ejemplo mostrará por consola la concatenación de todos y cada uno de los elementos del array. “unodostrescuatro”.

3.6.3. *B) La función para arrays reduce()*.

En el siguiente esquema se puede ver tanto el prototipo de la función reduce() como su aplicación al primer ejemplo del código anterior.

```
let numeros = [1, 2, 3, 4];
console.log (numeros.reduce((acc,v) =>{return acc+v;}));
```



3.6.3. *RECUERDA*.

Utiliza la función map() cuando tengas que realizar una operación por cada elemento de un array.

Utiliza la función reduce() cuando necesites realizar una operación por cada elemento de un array y haya que ir acumulando el resultado generado.

Utiliza la función arrow (=>) para funciones callback.

3.6.4. Funciones y métodos en objetos y vbls. string

JavaScript, al igual que otros lenguajes de programación, tienen muchos métodos para manejar strings:

```
var web = "prueba.com";
var longitud = web.length;
```

3.6.4. Funciones y métodos en objetos y vbls. string

Ahora, si queremos saber la posición (el lugar donde comienza) de la palabra “prueba” dentro del string web:

```
var web1= "La web prueba.com es una de las mejores en desarrollo";
var posicion = web1.lastIndexOf("prueba"); //posición valdrá 7
```

3.6.4. Funciones y métodos en objetos y vbls. string

Imagínese que la palabra o substring (no tiene por qué ser una palabra) se repite dentro del texto varias veces. Con el método lastIndexOf(), puede conocerse la posición de la última ocurrencia.

3.6.4. Funciones y métodos en objetos y vbls. string

- ***search(string1)***. Busca en un string un determinado valor (o expresión regular) y devuelve la posición donde la encontró.
- ***slice(inicio,fin)***. Extrae parte de un string y devuelve el nuevo string.
- ***substring(inicio,fin)***. Extrae parte de un string comenzando por la posición inicio y terminando en la posición fin.
- ***substr(inicio, longitud).*** Extrae caracteres de un string comenzando en la posición especificada en el primer parámetro con una longitud expresada en el segundo parámetro.

3.6.4. Funciones y métodos en objetos y vbls. string

- ***replace(string1, string2)***. Reemplaza en un string una cadena de caracteres (o una expresión regular) por la cadena expresada en el segundo parámetro. Esta función devuelve un string con los reemplazos realizados.
- ***toUpperCase()***. Convierte un string a mayúsculas.
- ***toLowerCase()***. Convierte un string a minúsculas.
- *concat(string1, string2)*. Concatena dos o más cadenas de caracteres devolviendo el string resultante.
- ***charAt(indice)***. Devuelve el carácter que se especifica en la posición expresada en el primer parámetro.

3.6.4. Funciones y métodos en objetos y vbls. string

- ***split(separador, límite)***. Crea un array de un string utilizando como separador el primer parámetro. El segundo parámetro es opcional y especifica el número máximo de elementos que puede tener dicho array. Pasado dicho número, JavaScript no creará nuevos elementos en el array.
- ***repeat(numVeces)***. Repite el string del cual se invoca el método el número de veces especificado en el primer parámetro.

3.6.4. Funciones y métodos en objetos y vbls. string

```
var stringSearch = "Hola Mundo";
console.log(stringSearch.search("Mun"));
//mostrará por consola 5

console.log(stringSearch.slice(2, stringSearch.length));
//mostrará por  consola "la Mundo"

console.log(stringSearch.substring(1, 4));
//mostrará por  consola "ola"

console.log(stringSearch.substr(1, 4));
//mostrará por consola "ola"
```

3.6.4. Funciones y métodos en objetos y vbls. string

```
console.log(stringSearch.replace( "a", "X"));
//mostrará por consola "Holx Mundo"

console.log(stringSearch.toUpperCase());
//mostrará por consola "HOLA MUNDO"

console.log(stringSearch.toLowerCase());
//mostrará por consola "hola mundo"
```

3.6.4. Funciones y métodos en objetos y vbls. string

```
var cad = "mmm";
console.log(stringSearch.concat(cad));
//mostrará por consola "Hola Mundommm"

console.log(stringSearch.charAt(5));
//mostrará por consola "M"

console.log(stringSearch.split(" "));
//Creará el Array ["Hola", "Mundo"]. Se ha utilizado como separador
//el espacio
console.log(stringSearch.repeat(2));
//mostrará por consola "Hola MundoHola Mundo"
```

3.6.5. Funciones globales del lenguaje JavaScript relativas a números

Para validar formularios o campos especiales, HTML5 tiene ciertas validaciones nativas, pero, en ocasiones, se necesita una validación más sofisticada. Para esos casos tenemos las siguientes funciones:

1. ***isFinite()***. Evalúa si un dato determinado es un valor finito. En caso positivo, devuelve *true* y, de lo contrario, devuelve *false*.

3.6.5. Funciones globales del lenguaje JavaScript relativas a números

1. ***isFinite()***. Evalúa si un dato determinado es un valor finito. En caso positivo, devuelve *true* y, de lo contrario, devuelve *false*.

```
var dato1 = isFinite(5/0); //devuelve false
var dato2 = isFinite(-3.14); //devuelve true
var dato3 = isFinite(0); //devuelve true
var dato4 = isFinite("5/9"); //devuelve false
var dato5 = isFinite("2018/12/05"); //devuelve false
var dato6 = isFinite("prueba.com"); //devueve false
```

3.6.5. Funciones globales del lenguaje JavaScript relativas a números

2. ***isNaN()***. Evalúa si un dato determinado no es numérico.

```
isNaN(4561);//devuelve false porque es un número
isNaN(0); //devuelve false
isNaN(-3.14); //devuelve false
isNaN(9+4); //devuelve false porque al evaluar la expresión el resultado es numérico.
isNaN('666');//devuelve false porque al evaluar el contenido el resultado es numérico.
isNaN('');//devuelve false.
isNaN(true);//devuelve false.
```

3.6.5. Funciones globales del lenguaje JavaScript relativas a números

```
isNaN(8/0); //devuelve false.
isNaN(0/0); //devuelve TRUE.
isNaN('Kursaal'); //devuelve TRUE al ser un string.
isNaN('2023/01/18');//devuelve TRUE al ser una fecha.
isNaN(undefined); //devuelve TRUE.
isNaN('NaN'); //devuelve TRUE.
isNaN(NaN); //devuelve TRUE.
```

3.6.5. Funciones globales del lenguaje JavaScript relativas a números

3. ***Number()***. Convierte un dato en número.

```
Number(true); //devuelve 1.
Number(false); //devuelve 0.
Number("666"); //devuelve 666.
```

4. ***parseFloat()***. Analiza un string y devuelve un número en coma flotante.

3.6.5. Funciones globales del lenguaje JavaScript relativas a números

4. ***parseFloat()***. Analiza un string y devuelve un número en coma flotante.

```
parseFloat("10"); //devuelve 10.
parseFloat("3.14"); //devuelve 3.14
parseFloat("   100"); //devuelve 100
parseFloat("80 gastos"); //devuelve 80.
```

3.6.5. Funciones globales del lenguaje JavaScript relativas a números

5. ***parseInt()***. Analiza un string y devuelve un número entero.

```
parseInt("10"); //devuelve 10.
parseInt("3.14"); //devuelve 3.
parseInt("   1000"); //devuelve 1000.
parseInt("80 gatos"); //devuelve 80.
```

3.6.5. Funciones globales del lenguaje JavaScript relativas a números

6. ***String()***. Convierte un dato en un string. Generalmente, se utiliza cuando se quiere convertir un dato numérico en un string.

```
String("666"); //devuelve 666.
String(3.14); //devuelve 3.14 convertido en string
```

3.6.6. Propiedades globales de JavaScript.

JavaScript tiene propiedades globales a todos los objetos creados:

- ***infinity***. Es un valor numérico que representa el infinito.

Ejemplo ***infinity.***

```
<!DOCTYPE html>
<html>
    <body>
        <p id="masInfinito"></p>
        <p id="menosInfinito"></p>
        <script>
            //1.7976931348623157E+308 es el límite de un número en coma flotante.
            document.getElementById("masInfinito").innerHTML = 1.8E+308;
            document.getElementById("menosInfinito").innerHTML = -1.8E+308;
        </script>
    </body>
</html>
```

3.6.7. Propiedades globales de JavaScript.

JavaScript tiene propiedades globales a todos los objetos creados:

- ***NaN***. La propiedad *NaN* representa un valor no numérico. En JavaScript, generalmente, se utiliza la función is*NaN()* vista anteriormente.
- ***undefined***. La propiedad *undefined* indica que a una variable no le ha sido asignado ningún valor (o que no ha sido declarada).

```
var x;
if (typeof x === "undefined"){
    y = "no definida";
}
```

3.6.7. Algunas funciones globales de JavaScript

JavaScript tiene funciones globales como las que se exponen a continuación:

- ***encodeURI()***. Codifica a URI. Codifica caracteres que no están permitidos en una URL como >, “ , [, etcétera.
- ***encodeURIComponent()***. Codifica una componente de URI. Codifica, además de los caracteres anteriores, otros como ?, =, :, etcétera.
- ***decodeURI()***. Decodifica una URI.
- ***decodeURIComponent()***. Decodifica una componente URI.

Ejemplos ***encodeURI/decodeURI***

```
var uri = "http://google.com";
var uriencoded = encodeURIComponent(uri);  //uriencoded contiene
http%3A%2F%2Fgoogle.com
var uridecoded = decodeURIComponent(uriencoded); //uridecoded contiene
http://google.com
```

3.6.7. Algunas funciones globales de JavaScript

- ***eval()***. Evalúa un string y lo ejecuta como si fuera código (un script).

```
var a = 2;
var b = 3;
var c = eval("a+b"); // c valdrá 5
```