JavaScript

Funciones

Funciones

Una función es ***un bloque de código que puede llamarse*** desde distintos puntos de un código para realizar una operación.

El uso de funciones tiene dos finalidades:

- ***No repetir*** el código de una tarea cada vez que la necesitemos.
- ***Estructurar*** el código de una tarea compleja en otras más sencillas.

Funciones

Declaración:

```
function nombre() {
      bloque de código  ( cuerpo de la función )
}
```

Llamada : *Una función sólo se ejecuta cuando es llamada*.

```
nombre();
```

Funciones

Ejemplo: Declaración y llamada de función *saludar()* que muestra un mensaje saludando al usuario

```
// Declaracion
function saludar() {
    alert("Hola!");
}

// Llamada -> Provoca la ejecucion del código de la función
saludar();
```

Una función únicamente se ejecuta cuando es llamada.

Parámetros y argumentos

Las funciones pueden recibir valores para hacer sus tareas (**parámetros**)

- Los parámetros se declaran separados por comas:

```
// Función que declara dos parámetros
function saludar( nombre, apellidos ) {
    alert(`Hola ${nombre} ${apellidos}`);
}
```

- Cada parámetro debe recibir un valor en la llamada a la función (**argumento**). Estos se asignan a los parámetros siguiendo el orden de declaración:

```
// Llamada a la funcion pasando dos argumentos
saludar("Pedro", "Gonzalez Perales");
```

Retorno de valores

Las funciones pueden retornar un valor como resultado.

- **Declaración** → La palabra clave ***return*** indica el valor retornado.

```
function identificador( parámetros ) {
        sentencias de la función  ( cuerpo de la función )
        return valor_de_retorno
}
```

- **Llamada** → La función se llama desde una asignación para obtener el valor retornado.

```
let valor = identificador( argumentos )
```

Retorno de valores

Ejemplo: El valor retornado por la función *media()* es obtenido en la llamada por la variable *resultado*:

```
// Declaracion funcion que retorna la media de dos valores
function media( a, b ) {
    return ( a + b ) / 2;       // Calculo y retorno del resultado
}

// Llamada obteniendo resultado
let resultado = media( 10, 20 );
console.log(resultado);             // Muestra 15
```

Convenciones sobre nombres y tareas.

Tareas realizadas:

- Debe implementar una única tarea
- Si es una tarea compleja debe implementarse llamando a otras funciones que ayuden a completar la tarea: (*divide y vencerás*).

Nombre de la función:

- Debe ser un verbo e identificar la tarea implementada.

  obtenerFecha()

  comprobarMail()

  mostrarDatos()

Ámbitos de variables

Las **variables locales** son *declaradas dentro del código de las funciones*.

```
// Declaracion
function mostrar_suma( a, b, c) {
    let r = a + b + c;      // variable local
}

// Llamada
mostrar_suma(10,20,30);
alert(`La suma es ${r}`);   // 'r' no existe
```

Las variables locales ***sólo existen dentro de la función*** y su uso fuera de la función da error:

```
▶ Uncaught ReferenceError: r is not defined                    ambito.js:8
    at ambito.js:8:21
```

Variables globales

Las **variables globales** son las *declaradas fuera de las funciones*.

```
// variable global
const IVA = 16;

// Declaracion
function calcular_importe( precio ) {
    return precio + (precio * ( IVA / 100 ));
}

// Llamada
let precio = calcular_importe( 1250.50 );
alert(`El importe es ${precio.toFixed(2)} con IVA ${IVA}%`);
```

Las variables globales suelen emplearse como constantes para contener valores empleados en múltiples partes de un script.

- Son *son accesibles* tanto fuera como dentro de las funciones.

Variables globales

Las **variables globales** *pueden modificarse dentro de las funciones*:

```
let x = 0;                              // Variable GLOBAL

function cambiarValor() {
    x = 10;                 // Modificacion valor de variable global
}

console.log("Antes funcion: " + x);         // Muestra 0
cambiarValor();
console.log("Despues funcion: " + x);       // Muestra 10
```

Funciones predefinidas

Estas son funciones ya definidas en Javascript y que pueden emplearse para ciertas comprobaciones y conversiones asociadas a números.

- ***parseInt(* cadena *)*** → Devuelve un valor entero equivalente a la cadena dada como argumento.
- ***parseFloat(* cadena *)*** → Devuelve el valor decimal equivalente a la cadena dada como argumento.

```
let cadena = "126.75";
let entero = parseInt( cadena );        // conversión a entero
let decimal = parseFloat( cadena );     // conversión a decimal
console.log(entero);
console.log(decimal);
```

```
126                                                    script.js:7
126.75                                                 script.js:8
```

parseInt(* cadena, base *) → Devuelve un valor entero equivalente a la cadena dada como argumento con el valor expresado en la base indicada.

```
// Conversion a enteros
console.log(parseInt("477", 8));        // Valor octal
console.log(parseInt("0110101", 2));    // Valor binario
console.log(parseInt("0xAAF", 16));     // Valor hexadecimal
```

```
319                                                    script.js:5
53                                                     script.js:6
2735                                                   script.js:7
```

Funciones predefinidas

Si *no es posible la conversión de cadena a número*, el resultado es *NaN* (*"Not A Number"*).

```
let cadena = "hola";
let entero = parseInt( cadena );        // conversión a entero
console.log(entero);
```

```
NaN                                                    script.js:6
```

- ***isNaN(valor)*** → Devuelve un valor lógico cierto si el valor dado es *NaN*. Se emplea para comprobar valores numéricos dados por el usuario mediante formularios o cuadros de diálogo.

```
let cadena = prompt("Indica un valor", ""); // Solicita valor por cuadro de dialogo de entrada
let decimal = parseFloat( cadena );         // conversión a entero
if ( isNaN( decimal ) == true ) {           // ¿Es NaN?
    alert("El valor dado no es un número")
} else {
    alert("El valor indicado es el " + decimal);
}
```

**** El formato de entrada de los decimales depende de la configuración regional del equipo en el que se ejecuta el navegador.***

Métodos predefinidos (*numeros*)

Son métodos que pueden llamarse a partir de valores primitivos numéricos para su manipulación:

- ***toFixed(*decimales*)*** → Redondea un valor decimal al número de decimales indicados.

```
let numero = 12.44;
console.log(numero.toFixed(1));     // Retorna 12.4
numero = 12.45
console.log(numero.toFixed(1));     // Retorna 12.4
numero = 12.46
console.log(numero.toFixed(1));     // Retorna 12.5
numero = 12.45
console.log(numero.toFixed(4));     // Retotrna 12.4500
```

(*) Si el nº de decimales indicado supera a los del valor, se añaden 0 a la derecha. Si es inferior se redondea la última cifra a la baja si es menor o igual que 5, o al alza si es superior.

Métodos predefinidos (*cadenas*)

Métodos de manejo de cadenas

toUpperCase()	Devuelve la cadena de texto en mayúsculas.
toLowerCase()	Devuelve la cadena de texto en minúsculas.
indexOf()	Devuelve la posición de la primera ocurrencia de un valor especificado en una cadena.
lastIndexOf()	Devuelve la posición de la última ocurrencia de un valor especificado en una cadena.
slice()	Extrae una sección de una cadena y devuelve una nueva cadena.
replace()	Reemplaza una parte de la cadena con otra.
split()	Divide una cadena en un array de subcadenas.
trim()	Elimina los espacios en blanco del principio y del final de una cadena.
includes()	Comprueba si una cadena está incluida en el contenido de otra
startsWith()	Comprueba si una cadena está incluida al principio de otra
endsWith()	Comprueba si una cadena está incluida al final de otra
repeat()	Retorna una cadena con la repetición de otra el número de veces indicado
charAt()	Retorna el carácter situado en la posición indicada de la cadena (0 el primero)

Propiedades de la cadenas

length	Propiedad que retorna el nº de caracteres de la cadena