Material descargable

egg

Teoría **JavaScript**

¡Hola! 👋 Te damos la bienvenida a JavaScript 3.0.

JavaScript, un lenguaje de programación versátil que nos permitirá agregar interactividad y funcionalidad a nuestras webs.

Nos adentraremos en este lenguaje abarcando los conceptos fundamentales y sus aplicaciones prácticas.

¡Que comience el viaje! 🚀

¿Qué es JavaScript?

JavaScript es un lenguaje de programación utilizado principalmente para hacer que las páginas web sean interactivas. Permite agregar funcionalidades a un sitio web, como botones que responden cuando se hace clic en ellos, formularios que validan la información ingresada por los usuarios y elementos que se actualizan automáticamente sin necesidad de recargar la página.

JavaScript se ejecuta en el navegador web y es muy popular debido a su facilidad de uso y versatilidad. Con JavaScript, los desarrolladores pueden crear experiencias dinámicas en línea para mejorar la interacción y la usabilidad de los sitios web.

¿Cuál es su historia?

JavaScript fue creado por Brendan Eich en 1995 para hacer las páginas web más interactivas. Originalmente llamado "LiveScript", se renombró como "JavaScript" para aprovechar la popularidad de Java en ese momento. Surgieron variantes como JScript de Microsoft, lo que llevó a la creación del estándar ECMAScript en 1997. A lo largo de los años, JavaScript ha evolucionado con nuevas versiones, como ECMAScript 6 en 2015. Hoy en día, JavaScript es ampliamente utilizado

tanto en el lado del cliente como en el lado del servidor, y ha transformado la forma en que interactuamos con las páginas web.

Sintaxis básica

En la programación, las variables desempeñan un papel fundamental, ya que nos permiten almacenar y manipular datos de manera eficiente. **Una variable en JavaScript es como un contenedor que guarda un valor específico**, como un número, una cadena de texto o cualquier otro tipo de información. Las variables nos permiten almacenar datos temporalmente en la memoria de la computadora para su posterior uso.

Al definir una variable en JavaScript, utilizamos la palabra clave var, let o const, seguida por un nombre que le asignamos a esa variable. También podemos asignarle un valor inicial opcional. El uso de var solía ser común, pero en la actualidad se prefiere el uso de let o const. La elección entre let y const depende de si deseamos reasignar valores a la variable o si queremos que tenga un valor constante e inmutable.

La variable nos proporciona flexibilidad en nuestro código, ya que podemos cambiar y actualizar su valor a medida que el programa se ejecuta. Esto es especialmente útil cuando necesitamos almacenar y trabajar con datos que pueden cambiar en diferentes etapas de un programa.

En resumen, las variables son esenciales en JavaScript y en la programación en general. Nos permiten almacenar y manipular datos de manera eficiente, lo que nos ayuda a crear programas más dinámicos y flexibles. El uso adecuado de variables nos permite organizar y controlar la información en nuestros programas, lo que resulta en un código más claro y legible.

Enlace

Existen tres formas principales de enlazar JavaScript con HTML:

1. **Enlace externo:**

Esta forma implica crear un archivo JavaScript separado con extensión ".js" y enlazarlo en el archivo HTML utilizando la etiqueta <script>.

El enlace externo se realiza mediante el atributo src, el cual especifica la ruta o URL del archivo JavaScript externo.

Por ejemplo: <script src="script.js"></script>.

Al utilizar esta forma, es importante asegurarse de que la ruta del archivo JavaScript sea correcta y esté accesible desde el archivo HTML.

2. **Enlace interno**:

Esta forma implica escribir el código JavaScript directamente dentro del archivo HTML utilizando la etiqueta <script>.

El código JavaScript puede colocarse en la sección <head> o en la sección <body> del archivo HTML.

```
<script>
  // Código JavaScript aquí
</script>
```

3. **Eventos en línea:**

Esta forma implica utilizar atributos de eventos en línea en elementos HTML para asociar directamente código JavaScript.

Los atributos de eventos en línea, como onclick, onload, onmouseover, etc., se utilizan para especificar el código JavaScript que se ejecutará cuando ocurra un evento determinado.

Por ejemplo: <button onclick="alert('¡Hiciste clic en el botón!')">Haz clic</button>.

Esta forma es útil para agregar interactividad rápida y sencilla a elementos HTML específicos.

Es importante recordar que estas tres formas de enlace permiten que el código JavaScript se ejecute en el contexto del navegador web cuando se carga o interactúa con el archivo HTML. Además, es recomendable colocar los enlaces externos e internos dentro de la sección <body> al final del archivo HTML para

asegurarse de que los elementos HTML se hayan cargado antes de que el código JavaScript intente manipularlos.

Recuerda que al enlazar JavaScript con HTML, es necesario asegurarse de utilizar la sintaxis correcta y verificar la ruta o ubicación de los archivos JavaScript para garantizar que se carguen y ejecuten correctamente en el contexto del archivo HTML.

Variables

Comencemos con los principios de Javascript y veamos la diferencia entre las variables.

¿var, let o const?

var: var se utilizaba anteriormente para declarar variables, pero desde la introducción de let y const, se ha vuelto menos común. Aún se encuentra en código heredado y puede tener comportamientos diferentes en relación con el ámbito (scope) de las variables. En la mayoría de los casos, es preferible utilizar let o const en su lugar.

let: let se utiliza para declarar variables que pueden ser reasignadas posteriormente. Es una buena elección cuando necesitas una variable cuyo valor puede cambiar durante la ejecución del programa. La variable declarada con let tiene un alcance de bloque, lo que significa que solo está disponible dentro del bloque donde se declara.

const: const se utiliza para declarar variables que tienen un valor constante e inmutable. Una vez que se le asigna un valor, no se puede reasignar. Es una buena elección cuando necesitas una variable cuyo valor no debe cambiar. Al igual que let, const también tiene un alcance de bloque.

La elección entre let y const depende de si necesitas que el valor de la variable pueda cambiar o no. Si sabes que el valor no cambiará a lo largo del programa, es preferible utilizar const para hacerlo explícito y evitar reasignaciones accidentales.

Es importante considerar la legibilidad y el alcance de las variables al elegir qué palabra clave utilizar. Opta por utilizar let o const siempre que sea posible, ya que ofrecen un comportamiento más predecible y facilitan el mantenimiento del código. Además, al utilizar const, estás indicando tu intención de que el valor de la variable no debe cambiar, lo cual es útil para comprender el código en el futuro.

Recuerda que el uso adecuado de var, let y const puede ayudarte a escribir un código más limpio y evitar problemas relacionados con el ámbito y la reasignación accidental de variables.

Entrada y salida de datos

Una consola web es una herramienta que se utiliza principalmente para registrar información asociada a una página web como: solicitudes de red, JavaScript, errores de seguridad, advertencias, CSS, etc. Esta, nos permite interactuar con una página web ejecutando una expresión JavaScript en el contenido de la página. En JavaScript, Console, es un objeto que proporciona acceso a la consola de depuración del navegador. Podemos abrir una consola en el navegador web Chrome, usando: Ctrl + Shift + J para Windows/Linux y Option + Command ⌘ + J para Mac.

El objeto “console” nos proporciona varios métodos diferentes, como:

- log()
- error()
- warn()
- clear()
- time() y timeEnd()
- table()
- count()
- group() y groupEnd()
- custom console logs

Método	Descripción	Ejemplo

console.log()	Se utiliza principalmente para registrar (imprimir) la salida en la consola. Podemos poner cualquier tipo dentro del log (), ya sea una cadena, matriz, objeto, booleano, etc.	console.log("abc");
		console.log(123);
		console.log([1,2,3,4]);

console.error() **console.warn**	Error: se utiliza para registrar mensajes de error en la consola. Warn: se usa para registrar mensajes de advertencia	console.error("Mensaje de error"); console.warn("Mensaje de advertencia");
console.clear()	Se usa para limpiar la consola.	console.clear();
console.time() **console.timeEnd()**	Siempre que queramos saber la cantidad de tiempo empleado por un bloque o una función, podemos hacer uso de los métodos time() y timeEnd().	console.time('abc'); console.timeEnd('abc');
console.table()	Este método nos permite generar una tabla dentro de una consola. La entrada debe ser una matriz o un objeto que se mostrará como una tabla.	console.table({'a':,, 'b':2});

console.count()	Este método se usa para contar el número que la función alcanza con este método de conteo.	for(let i=0;i<5;i++){
		console.count(i);
		}

Objeto window

El objeto window de Javascript nos sirve para controlar la ventana del navegador. Es el objeto principal en la jerarquía y contiene las propiedades y métodos para controlar la ventana del navegador. Tiene algunos métodos como:

Salida de datos:

Para mostrar información al usuario, podemos utilizar el método console.log(). Este método imprime un mensaje o valor en la consola del navegador. Por ejemplo:

```
console.log("Hola, mundo!");
```

También podemos utilizar el método alert() para mostrar mensajes emergentes en una ventana de diálogo. Por ejemplo:

```
alert("¡Bienvenido!");
```

Además, podemos manipular el contenido de una página web utilizando el método document.write(). Este método es útil cuando deseamos mostrar información directamente en el cuerpo del documento HTML. Por ejemplo:

```
document.write("El resultado es: " + resultado);
```

Entrada de datos:

Para obtener datos del usuario, podemos utilizar el método prompt(). Este método muestra un cuadro de diálogo que permite al usuario ingresar un valor. El valor ingresado por el usuario se almacena en una variable. Por ejemplo:

```
var nombre = prompt("Ingrese su nombre:");
```

También podemos utilizar formularios HTML para obtener datos del usuario. Los datos ingresados en los campos de un formulario se pueden recuperar utilizando JavaScript. Para ello, podemos acceder a los elementos del formulario mediante su identificador o mediante métodos como getElementById().

Por ejemplo:

HTML:

```
<input type="text" id="nombreInput">
<button onclick="obtenerNombre()">Enviar</button>
```

JavaScript

```
function obtenerNombre() {
  var nombre = document.getElementById("nombreInput").value;
  console.log("Hola, " + nombre + "!");
}
```

Variables

Una variable es un contenedor para almacenar valores en JavaScript. Permite guardar y manipular datos durante la ejecución de un programa. Para declarar una variable en JavaScript, utilizamos las palabras clave var, let o const, seguidas por el nombre de la variable. Por ejemplo:

```
var edad;
let nombre;
const PI = 3.1416;
```

La palabra clave var se utilizaba en versiones antiguas de JavaScript, pero ahora es más común utilizar let y const.

→ **let** se utiliza para declarar variables que pueden ser reasignadas a diferentes valores.

→ **const** se utiliza para declarar variables cuyo valor no puede cambiar una vez asignado.

Tipos de variables y operaciones:

- **Variables numéricas:**

  Las variables numéricas almacenan valores numéricos, como enteros o decimales. Podemos realizar operaciones aritméticas básicas con ellas, como suma, resta, multiplicación y división. Por ejemplo:

```
let edad = 25;
let altura = 1.75;

let suma = edad + altura;
let resta = edad - altura;
let multiplicacion = edad * altura;
let division = edad / altura;
```

- **Variables de cadena de texto (string):**

  Las variables de tipo string almacenan texto. Podemos concatenar cadenas de texto utilizando el operador +. También podemos utilizar métodos de cadenas para manipular y obtener información sobre ellas. Por ejemplo:

```
let nombre = "Juan";
let apellido = "Pérez";

let nombreCompleto = nombre + " " + apellido;
let longitudNombre = nombre.length;
let mayusculas = apellido.toUpperCase();
```

- **Variables booleanas:**

  Las variables booleanas almacenan valores de verdad, que pueden ser true (verdadero) o false (falso). Podemos utilizar operadores lógicos como && (y), || (o) y ! (negación) para realizar operaciones booleanas. Por ejemplo:

```
let esMayorDeEdad = true;
let tieneLicencia = false;

let puedeConducir = esMayorDeEdad && tieneLicencia;
let noPuedeVotar = !esMayorDeEdad || tieneLicencia;
```

Template strings

Las template strings, o cadenas de texto de plantilla, son una de las herramientas de ES6 para trabajo con cadenas de caracteres que nos pueden venir muy bien para producir un código Javascript más claro. Usarlas es por tanto una recomendación dado que facilitará el mantenimiento de los programas, gracias a que su lectura será más sencilla con un simple vistazo del ojo humano.

Concatenación de Variables

En un programa realizado en JavaScript, y en cualquier lenguaje de programación en general, es normal crear cadenas en las que tenemos que juntar cadenas con los valores tomados desde las variables.

```
var sitioWeb = "DesarrolloWeb.com";

var mensaje = 'Bienvenido a ' + sitioWeb;
```

Eso es muy fácil de leer, pero a medida que el código se complica y en una cadena tenemos que concatenar el contenido de varias variables, el código comienza a ser más enrevesado.

```
var nombre = 'Miguel Angel';

var apellidos = 'Alvarez';

var profesion = 'desarrollador';

var perfil = ' ' + nombre + ' ' + apellidos + ' es ' + profesion;
```

Quizás estás acostumbrado a ver esto así. El código está bien y no tiene ningún problema, pero podría ser mucho más bonito si usas los template strings.

Crear un Template String

Para crear un template string simplemente tienes que usar un carácter que se usa poco, como apertura y cierre de la cadena. Es el símbolo del acento grave. (`)

```
var cadena = `Esto es un template String`;
```

Usos de los Template Strings

Los template strings tienen varias características interesantes que, cómo decíamos, facilitan la sintaxis. Veremos a continuación algunos de ellos con código de ejemplo.

Concatenación de Valores

Creo que lo más interesante es el caso de la concatenación que genera un código poco claro hasta el momento. Echa un vistazo al código siguiente que haría lo mismo que el que hemos visto anteriormente del perfil.

```
var nombre = 'Miguel Angel';

var apellidos = 'Alvarez' ;

var profesion = 'desarrollador';

var perfil = `<b>${nombre} ${apellidos}</b> es ${profesion}`;
```

Como puedes comprobar, dentro de un template string es posible colocar expresiones encerradas entre llaves y precediendo de un símbolo "$". Algo como ${ expresion }.

En las expresiones podemos colocar código que queramos volcar, dentro de la cadena. Las usamos generalmente para colocar valores de variables, pero también servirían para colocar operaciones matemáticas, por ejemplo.

```
var suma = `45 + 832 = ${45 + 832}`;
```

O bien algo como esto:

```
var operando1 = 7;

var operando2 = 98;

var multiplicacion = `La multiplicación entre ${operando1} y ${operando2}
equivale a ${operando1 * operando2}`;
```

Saltos de línea dentro de cadenas

Hasta ahora, si queremos hacer una cadena con un salto de línea teníamos que usar el carácter de escape "contrabarra n".

```
var textoLargo = "esto es un texto\ncon varias líneas";
```

Con un template string tenemos la alternativa de colocar el salto de línea tal cual en el código y no producirá ningún problema.

```
var textoLargo = `esto es un texto

con varias líneas`;
```

Tipos de datos en javascript

Los tipos de datos JavaScript se dividen en dos grupos: tipos primitivos y tipos objeto.

Primitivos	
Dato	**Descripción**
Numérico	Números, ya sea enteros o reales.
String	Cadenas de texto.
Boolean	Valores lógicos como true o false.
null	Cuando un dato no existe.
undefined	Cuando no se le asigna un valor a la variable.

Los tipos objeto tienen sus propias subdivisiones

Objetos	
Tipo De Objeto	**Descripción**
Tipo predefinido de JavaScript	Date: fechas
Tipo predefinido de JavaScript	RegExp: expresiones regulares
Tipo predefinido de JavaScript	Error: datos de erros
Tipos definidos por el programador / usuario	Funciones Simples y Clases
Arrays	Serie de elementos o formación tipo vector o matriz. Lo consideramos un objeto especial

Typeof

La función ***typeof*** se utilizada para obtener el tipo de dato que tiene una variable.
console.log(typeof 42);

```
// expected output: "number"

console.log(typeof 'blubber');

// expected output: "string"

console.log(typeof true);

// expected output: "boolean"
```

Condicionales en Javascript

Al igual que en Java, existen los condicionales que nos van a ayudar a modificar el flujo de ejecución del programa.

IF

El condicional if es un condicional lógico que evalúa el camino a tomar en base a la evaluación de una condición. Supongamos el siguiente ejemplo, mi sobrino quiere subirse a una montaña rusa, pero para ello tiene que aprobar las dos siguientes condiciones: tener más de 18 años y medir más de 160 cm. La evaluación de esas dos condiciones da por verdadero se podrá subir de lo contrario no podrá.

```
let edad = 15;

let altura = 166;

if(edad>18 && altura>160){

 console.log("Puedes subirte :D");

}else{

 console.log("No te podes subir”);

}
```

Como se puede ver, si la condición a evaluar se cumple, es decir, da verdadero, mostrará el mensaje *"Puedes subirte :D"*, en caso de que de falso mostrará *"No te puedes subir "*. **Por otra parte, JavaScript permite también agregar la condición else if**

```
if(a == 2){
 console.log("a es igual a 2");
}else if(a < 2){
 console.log("a es menor que 2");
}else{
 console.log("a es mayor que 2");
}
```

IF Ternario

El if ternario nos permite resolver en una línea una expresión lógica asignando un valor. Proviene del lenguaje C, donde se escriben muy pocas líneas de código y donde cuanto menos escribamos más elegantes seremos. Este operador es un claro ejemplo de ahorro de líneas y caracteres al escribir los scripts. Lo veremos rápidamente, pues la única razón que lo veamos es para que sepan que existe y si lo encuentran en alguna ocasión sepan identificarlo y cómo funciona.

```
Variable = (condición) ? valor1 : valor2
```

Este ejemplo no solo realiza una comparación de valores, además asigna un valor a una variable. Lo que hace es evaluar la condición (colocada entre paréntesis) y si es positiva asigna el valor1 a la variable y en caso contrario le asigna el valor2.

Veamos un ejemplo:

```
momento = (hora_actual < 12) ? "Antes del mediodía" : "Después del mediodía"
```

SWITCH

La declaración switch evalúa una expresión, **comparando el valor de esa expresión con una instancia case,** y ejecuta declaraciones asociadas a ese case, así como las declaraciones en los case que siguen.

El programa primero busca la primera instancia case cuya expresión se evalúa con el mismo valor de la expresión de entrada (usando comparación estricta, ===) y luego transfiere el control a esa cláusula, ejecutando las declaraciones asociadas. Si no se encuentra una cláusula de case coincidente, el programa busca la cláusula default opcional, y si se encuentra, transfiere el control a esa instancia, ejecutando las declaraciones asociadas.

Al igual que Java, la declaración break es opcional y está asociada con cada etiqueta de case y asegura que el programa salga del switch una vez que se ejecute la instrucción coincidente y continúe la ejecución en la instrucción siguiente. Si se omite el break, el programa continúa la ejecución en la siguiente instrucción en la declaración de switch.

```
switch (expr) {
case 'Naranjas':
 console.log ('El kilogramo de naranjas cuesta
$0.59.'); break;
 case 'Mangos':
   case 'Papayas':
 console.log('El kilogramo de mangos y papayas cuesta
$2.79.'); break;
 default:
 console.log('Lo lamentamos, por el momento no disponemos de ' +
expr + '.');
}
```

Estructuras Repetitivas

Veamos cómo son las estructuras repetitivas en JavaScript.

WHILE

Crea un bucle que ejecuta una sentencia especificada mientras cierta condición se evalúe como verdadera. Dicha condición es evaluada antes de ejecutar la sentencia

```
let a = 0;
while(a != 10){
 console.log(++a);
}
```

DO WHILE

La sentencia (hacer mientras) crea un bucle que ejecuta una sentencia especificada, hasta que la condición de comprobación se evalúa como falsa. La condición se evalúa después de ejecutar la sentencia, dando como resultado que la sentencia especificada se ejecute al menos una vez.

```
let a = 0;
do{
 console.log(++a);
}while(a!=10);
```

FOR

El bucle FOR se utiliza para repetir una o más instrucciones un determinado número de veces. De entre todos los bucles, el FOR se suele utilizar cuando sabemos seguro el número de veces que queremos que se ejecute. La sintaxis del bucle for va a ser la misma que en Java:

```
for ([expresion-inicial]; [condicion]; [expresion-final]){ }
for(let i = 0; i < 10; i++){
 console.log("El valor de i es " + i);
}
```

BREAK

Termina el bucle actual, sentencia switch o label y transfiere el control del programa a la siguiente sentencia.

```
for (let i = 0; i < 10; i++) {
 if(i == 5){
 break;
 }
 console.log("Estamos por la vuelta "+i);
}
```

CONTINUE

Termina la ejecución de las sentencias de la iteración actual del bucle actual o la etiqueta y continua la ejecución del bucle con la próxima iteración.

En contraste con la sentencia break, continue no termina la ejecución del bucle por completo; en cambio,

- **En un bucle while, salta de regreso a la condición.**
- **En un bucle for, salta a la expresión actualizada.**

La sentencia *continue* puede incluir una etiqueta opcional que permite al programa saltar a la siguiente iteración del bucle etiquetado en vez del bucle actual. En este caso, la sentencia *continue* necesita estar anidada dentro de esta sentencia etiquetada.

```
for (let i = 0; i < 10; i++) {
 if(i == 5){
 continue;
}
```

```
console.log("Estamos por la vuelta " + i);
}
```

LABEL

Proporciona a una sentencia con un identificador al que se puede referir al usar las sentencias break o continue. Por ejemplo, puede usar una etiqueta para identificar un bucle, y entonces usar las sentencias break o continue para indicar si un programa debería interrumpir el bucle o continuar su ejecución.

label o etiqueta: sentencia

Ejemplo:

```
exterior: for (let i = 0; i < 10; i++) {
 for (let j = 0; j < 10; j++) {
 if(i == 4 && j == 4){
 console.log("Vamos a cortar ambos for");
 break exterior;
 }
console.log(i+j+10*i);
}}
```

Funciones en Javascript

Una función, en JavaScript es, al igual que Java, como una serie de instrucciones que englobamos dentro de un mismo proceso. Este proceso se podrá luego ejecutar desde cualquier otro sitio con solo llamarlo. Por ejemplo, en una página web puede haber una función para cambiar el color del fondo y desde cualquier punto de la página podríamos llamarla para que nos cambie el color cuando lo deseemos.

La sintaxis de una función en JavaScript es la siguiente:

```
function nombrefuncion (){
 instrucciones de la función
 ...
}
```

Primero se escribe la palabra function, reservada para este uso. Seguidamente, se escribe el nombre de la función, que como los nombres de variables puede tener números, letras y algún carácter adicional como en guion bajo y los paréntesis donde irán nuestros parámetros. A continuación se colocan entre llaves las distintas instrucciones de la función. Las llaves en el caso de las funciones no son opcionales, además es útil colocarlas siempre como se ve en el ejemplo, para que se reconozca fácilmente la estructura de instrucciones que engloba la función.

Un ejemplo de función que escribe hola mundo en la consola sería:

```
function holaMundo(){
console.log('Hola Mundo');
}
```

Después, para llamar a esta función, hay que invocarla mediante su nombre holaMundo();

Dónde colocamos las funciones Javascript

En principio, podemos colocar las funciones en cualquier parte de la página, siempre entre etiquetas **<script>**. No obstante, existe una limitación a la hora de colocarla con relación a los lugares desde donde se la llame. Usualmente, lo más fácil es colocar la función antes de cualquier llamada a la misma y así seguro que nunca nos equivocaremos.

Existen dos opciones posibles para colocar el código de una función:

a) Colocar la función en el mismo bloque de script: En concreto, la función se puede definir en el bloque <script> donde esté la llamada a la función, aunque es

indiferente si la llamada se encuentra antes o después del código de la función, dentro del mismo bloque <script>.

```
<script>
miFuncion();
function miFuncion(){
 //hago algo...
 console.log("Esto va bien");
}
</script>
```

b) Colocar la función en otro bloque de script: También es válido que la función se encuentre en un bloque <SCRIPT> anterior al bloque donde está la llamada.

```
<html>
<head>
 <title>MI PÁGINA</title>
<script>
function miFuncion(){
 //hago algo...
 console.log("Esto va bien");
}
</script>
</head>
<body>
<script>
miFuncion()
</script>
```

```
</body>

</html>
```

Parámetros de las Funciones

Los parámetros se usan para mandar valores a las funciones. Una función trabajará con los parámetros para realizar las acciones. Por decirlo de otra manera, **los parámetros son los valores de entrada que recibe una función.**

Los parámetros en JavaScript, son iguales que en Java, la única diferencia es que, a diferencia de Java, no tenemos que especificar el tipo de dato que recibe la función, sino que va a ser el tipo de dato que enviemos como parámetro.

```
function escribirBienvenida(nombre){

console.log('Hola ' + nombre);

}

escribirBienvenida('Agustin');

// O podemos hacerlo con una variable

let nombre = 'Agustin';

escribirBienvenida(nombre);
```

Devolver valores en Funciones

Las funciones en Javascript también pueden retornar valores. De hecho, ésta es una de las utilidades más esenciales de las funciones.

Veamos un ejemplo de función que calcula la media de dos números. La función recibirá los dos números y retornará el valor de la media.

```
function media(valor1,valor2){

 let resultado;

 resultado = (valor1 + valor2) / 2;
```

```
    return resultado;
}
```

Para especificar el valor que retornará la función se utiliza la palabra *return* seguida del valor que se desea devolver. En este caso se devuelve el contenido de la variable resultado, que contiene la media calculada de los dos números.

Pero, cómo podemos recibir un dato que devuelve una función. Cuando una función devuelve un valor, simplemente se sustituye la llamada a la función por ese valor que devuelve. Así pues, para almacenar un valor de devolución de una función, tenemos que asignar la llamada a esa función como contenido en una variable, y eso lo haríamos con el operador de asignación =.

```
let miMedia

miMedia = media(12,8);

console.log(miMedia);
```

Funciones Flecha

Hay otra sintaxis muy simple y concisa para crear funciones, que a menudo es mejor que las expresiones de funciones.

Se llama *"funciones de flecha"*, porque se ve así:

```
let func = (arg1, arg2, ..., argN) => expresion
```

Esto crea una función func (nombre de la función) que acepta parámetros arg1..argN, luego evalúa la expresion de la derecha con su uso y devuelve su resultado.

En otras palabras, es la versión más corta de:

```
let func = function(arg1, arg2, ..., argN) {

return expresion;

};
```

Veamos un ejemplo completo:

```
let sum = (a, b) => a + b;

/* Esta función de flecha es una forma más corta de:

let sum = function(a, b) {

 return a + b;

};

*/

console.log(sum(1, 2)); // 3
```

Como puedes ver (a, b) => a + b significa una función que acepta dos parámetros llamados a y b. Tras la ejecución, evalúa la expresión a + b devuelve el resultado.

Si solo tenemos un argumento, se pueden omitir paréntesis alrededor de los parámetros, lo que lo hace aún más corto.

```
let double = n => n * 2;
```

Esto sería equivalente a escribir:

```
 let double = function(n) { return n * 2 }

console.log(double(3)); //6
```

Si no hay parámetros, los paréntesis estarán vacíos (pero deben estar presentes):

```
let saludar = () => console.log("Hola!");

 saludar();
```

Este tema lo hemos visto para que sepan que existe y si lo encuentran en alguna ocasión sepan identificarlo y cómo funciona. Recomendamos que antes de pasar a las funciones flecha, aprendamos a trabajar las funciones “normales”.