ISTPET
INSTITUTO TRAVERSARI

JAVASCRIPT

Cristian Andrés Cola Pérez
Trujillo Quinto Wilfrido Almicar
Josué Emanuel Tello Montero
Francisco Xavier Mesías Flores

Cristian Andrés Cola Pérez
Wilfrido Amilcar Trujillo Quinto
Josué Emanuel Tello Montero
Francisco Xavier Mesías Flores

JAVASCRIPT

JAVASCRIPT

Cristian Andrés Cola Pérez
Wilfrido Amilcar Trujillo Quinto
Josué Emanuel Tello Montero
Francisco Xavier Mesías Flores

Javascript

Javascript

Autores:

Cristian Andrés Cola Pérez
Instituto Superior Tecnológico "Mayor Pedro Traversari"
Carrera de Desarrollo de Software
cristian.cola@istpet.edu.ec
https://orcid.org/0009-0007-5337-5702

Josué Emanuel Tello Montero
Instituto Superior Tecnológico "Mayor Pedro Traversari"
Carrera de Desarrollo de Software
josue.tello@istpet.edu.ec
https://orcid.org/0009-0006-6554-7244

Francisco Xavier Mesías Flores
Instituto Superior Tecnológico "Mayor Pedro Traversari"
Carrera de Desarrollo de Software
francisco.mesias@istpet.edu.ec
https://orcid.org/0009-0000-7484-5450

Primera edición, julio 2024
JavaScript

ISBN: 978-9942-607-75-1 (eBook)
ISSN: 2600-5719 (electronic)

Editado por:
Centro de Investigación y Desarrollo Profesional

Babahoyo - Ecuador
Móvil - (WhatsApp): (+593) 9 8 52-92-824
www.cidepro.org
E-mail: **editorial@cidepro.org**

Este texto ha sido sometido a un proceso de evaluación por pares externos con base en la normativa editorial de CIDEPRO.

Diseño y diagramación:
CIDEPRO Editorial

Diseño, montaje y producción editorial:
CIDEPRO Editorial

Hecho en Ecuador
Made in Ecuador

ÍNDICE

INTRODUCCIÓN

JavaScript es uno de los lenguajes de programación más populares y versátiles en el desarrollo web. Es una herramienta esencial para cualquier persona interesada en crear experiencias interactivas y dinámicas en sitios web y aplicaciones. Con JavaScript, puedes manipular los elementos de una página, responder a eventos del usuario y realizar cálculos complejos. Además, puedes crear juegos, animaciones 2D y gráficos 3D, aplicaciones integradas basadas en bases de datos entre otras.

Este libro es una guía práctica para aprovechar todo el potencial de JavaScript en el diseño de interfaces web. A través de ejemplos claros y proyectos prácticos, aprenderás a crear interacciones fluidas y atractivas que cautiven a los usuarios. Además, profundizarás en los conceptos esenciales de JavaScript, desde los fundamentos básicos hasta técnicas más avanzadas. También, se explorará el uso de bibliotecas y frameworks populares, como jQuery, que facilitan el desarrollo y mejoran la eficiencia en la creación de interfaces web interactivas. El principal objetivo es desarrollar la confianza y habilidad necesarias para enfrentar distintos desafíos en el diseño de interfaces web.

ISTPET
INSTITUTO TRAVERSARI

CAPÍTULO 1

Primeros pasos con el lenguaje JavaScript

CAPÍTULO 1

Primeros pasos con el lenguaje JavaScript

1.1. Lenguaje de programación

Los lenguajes de programación son un conjunto de reglas, símbolos y sistemas que se utilizan para comunicarse con las computadoras. Estos lenguajes permiten a los programadores escribir instrucciones que las computadoras pueden entender y ejecutar. Los programas informáticos creados con estos lenguajes pueden ser utilizados para realizar una amplia variedad de tareas, desde el procesamiento de texto y la creación de gráficos hasta la navegación por Internet y los juegos.

Hay muchos lenguajes de programación diferentes, cada uno con sus propias ventajas y desventajas. El mejor lenguaje de programación para una tarea determinada dependerá de las necesidades específicas de un proyecto de software. Por ejemplo, si se está creando una aplicación web, Java, JavaScript o Python pueden ser buenas opciones.

1.2. El lenguaje JavaScript

JavaScript es un lenguaje de programación utilizado para crear diferentes efectos en páginas web y delimitar qué tipo de participación con el usuario tendrá la página, se ejecuta en el navegador del cliente. JavaScript es un lenguaje de programación bastante sencillo, incluso para las personas que no tienen experiencia programando.

JavaScript es un lenguaje de programación muy versátil que se puede utilizar para crear una amplia variedad de efectos, como animaciones, cambios y transiciones de color y movimiento. Además, se puede utilizar para crear páginas web interactivas y aplicaciones web. JavaScript es un lenguaje muy potente que permite la programación de pequeños scripts, así como de programas más robustos y complejos. Al aprender JavaScript, se puede controlar el flujo del código y colocar scripts para dar solución a distintas necesidades.

Diferencias entre JavaScript y Java

JavaScript y Java son dos lenguajes de programación diferentes, a pesar de que sus nombres son similares y comparten algunas similitudes, como la sintaxis. JavaScript es un lenguaje de programación que se ejecuta sobre todo en el lado del cliente, mientras que Java es un lenguaje de programación compilado que se ejecuta en el lado del servidor.

JavaScript se utiliza principalmente para agregar interactividad a las páginas web, mientras que Java se puede utilizar para crear otro tipo de aplicaciones en las cuales no se profundizará.

A continuación, se presentan algunas de las principales diferencias entre JavaScript y Java:

- **Interpretado vs. compilado:** JavaScript es un lenguaje interpretado, lo que significa que se traduce a código máquina en el momento de la ejecución. Java es un lenguaje compilado, lo que significa que se traduce a código máquina antes de la ejecución.
- **Lado del cliente vs. lado del servidor:** JavaScript se ejecuta sobre todo en el lado del cliente, que es el navegador web del usuario. Java se puede ejecutar en el lado del cliente o del servidor.
- **Usos:** JavaScript se utiliza principalmente para agregar interactividad a las páginas web. Java se puede utilizar para crear una amplia variedad de aplicaciones, incluyendo aplicaciones de escritorio, aplicaciones móviles y aplicaciones web.

Aunque JavaScript y Java son dos lenguajes diferentes, ambos tienen sus propias fortalezas y debilidades. JavaScript es un lenguaje más fácil de aprender y usar que Java.

1.3. Porque el lenguaje importa

JavaScript se puso en marcha en el año en 1995, como una estructura para agregar programas en páginas web al navegador Netscape Navigator**.** Desde entonces ha sido adoptado por los otros navegadores web. Gracias a JavaScript las aplicaciones web modernas han sido creadas, dentro de estas se encuentran**:** aplicaciones con las que puedes interactuar de manera directa, sin recargar la página para cada acción a realizar. JavaScript también es utilizado en sitios web más tradicionales para suministrar diversas formas de interactividad.

El lenguaje de programación es una de las decisiones más importantes que los desarrolladores deben tomar al iniciar un proyecto de software. La elección del lenguaje puede tener un impacto significativo en el desarrollo, el rendimiento, la productividad y la experiencia como desarrolladores.

Rendimiento y eficiencia

Cada lenguaje tiene niveles de rendimiento y eficiencia diferente. Los lenguajes de bajo nivel, como C y C++, tienden a ser más rápidos y eficientes en el uso de recursos del sistema, lo que los hace más adecuados para aplicaciones que requieren un alto rendimiento. Por otro lado, los lenguajes de alto nivel, como Python y JavaScript, ofrecen mayor productividad y facilidad de desarrollo.

El lenguaje de programación web es una pieza fundamental en el proceso de desarrollo de software y su elección tiene implicaciones en el rendimiento, la productividad, la escalabilidad y la mantenibilidad del proyecto. Evaluar cuidadosamente las necesidades y objetivos del proyecto, así como las características y capacidades con JavaScript, es crucial para tomar una decisión adecuada en el desarrollo del software.

1.4. Sintaxis

La sintaxis se define como un conjunto de reglas que deben seguirse al escribir el código fuente del programa para considerarse como correcto.

Instrucción

La sintaxis de JavaScript es simple. Las instrucciones se separan generalmente por un punto y coma, que se coloca al final de cada instrucción. Por ejemplo:

```
sentencia_1;
sentencia_2;
sentencia_3;
sentencia_n;
```

En realidad, el punto y coma no es necesario si la instrucción está en la línea posterior como en el ejemplo.

Sin embargo, si se escribe varias instrucciones en una sola línea, como en el siguiente ejemplo, el punto y coma es obligatorio.

```
sentencia_1; sentencia_2
sentencia_3; sentencia_n
```

Compresión de scripts

Algunas secuencias de comandos están disponibles en un formato comprimido, es decir, todo el código se escribe en secuencia, no hay retornos de línea. Esto reduce considerablemente el tamaño de una secuencia de comandos y sirve para asegurar que la página se cargue rápidamente.

Espacios JavaScript no es sensible a los espacios. Esto significa que puede alinear las instrucciones que desee, siempre que no interfiera con la secuencia de comandos. Por ejemplo:

```
instruccion_1;
        instruccion_1_1;
        instruccion_1_2;
instruccion_2;
        instruccion2_1;
```

Sangría y presentación

La sangría en la programación es una manera de estructurar el código para hacerlo más legible. Las instrucciones van en varios niveles y espacios para crear una jerarquía. Un ejemplo de código:

```
funcion ejemplo(ejemID) {
        var ejem = document.getElementById(ejemID);
        if (ejem.style.display == 'block') {
                ejem.style.display = 'none';
        } else {
                ejem.style.display = 'block';
        }
}
```

La presentación de los códigos también es importante. En el código mostrado anteriormente, se puede ver que hay espacios para “airear” todo el código y sólo hay una declaración por línea (salvo if else). En algunos casos se puede escribir el código como en el ejemplo:

```
function ejemplo(ejemID){
        var ejem=document.getElementById(ejemID);
        if(ejem.style.display=='block'){
                ejem.style.display='none';
        }else{
                ejem.style.display='block';
                }
}
```

Comentarios

Los comentarios son observaciones realizadas para explicar el funcionamiento de un script, una instrucción o incluso un grupo de instrucciones. Los comentarios no interfieren con la ejecución de un script. Hay dos tipos de comentarios: los de fin de línea y los multilínea. Comentarios de fin de línea. Se utilizan para comentar una instrucción. Comienza con dos barras de división:

```
sentencia_1 //Esta es mi primera instrucción
sentencia_2//La tercera declaración es la siguiente:
sentencia_3;
```

Comentarios multilínea: Este tipo permite saltos de línea. Un comentario multilínea comienza con / * y termina con * /:

```
/ * Este script consta de tres pasos:
- Instrucción uno está haciendo algo
- Instrucción dos para otra cosa
 - Instrucción tres que pone fin a la secuencia de comandos
* /
sentencia_1;
sentencia_2;
sentencia_3 / / Fin del script
```

Funciones

En el ejemplo, se utilizó la función alert (). Una función consta en dos partes: su nombre, seguido por un par de paréntesis (una apertura y un cierre):

```
primeraFuncion ()
```

En el paréntesis se indican los argumentos también llamados parámetros. Estos contienen los valores que se trasladan a la función. En el caso de ¡Hola mundo!, Son las palabras "¡Hola, mundo! " lo que se transfieren como parámetros:

```
alert ('Hola mundo!');
```

Diferentes formas de declara funciones en JavaScript

- **Declaración de la función:** Esta es una de las formas más conocidas de declarar en una función que sirve para la mayoría de los lenguajes de programación. La declaración de función se realiza utilizando la palabra clave function, seguida de un nombre de función obligatorio, una lista de parámetros entre paréntesis y un par de llaves que delimitan el cuerpo del código.

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

- **Expresión de una función:** La expresión de función es similar a la declaración de función, pero en lugar de declarar una función con un nombre, se asigna una función anónima a una variable.

```
const sum = function(a, b) {
    return a + b
}
```

- **Función de flecha:** Las funciones de flecha son una característica relativamente nueva en JavaScript que se introdujo en la versión ES6. Son una forma más concisa de escribir funciones y se utilizan comúnmente en el desarrollo web moderno. Se definen utilizando la sintaxis () => {}. La sintaxis es más corta que la de las funciones regulares y se puede utilizar para definir funciones anónimas o con nombre. Además, las funciones de flecha tienen un comportamiento especial en cuanto a cómo manejan el valor de this.

  Forma 1: Forma para retornar un valor

```
const sum = (a, b) => {
    return a + b
};
```

Forma 2: Se está retornando igual el valor, pero se lo puede simplificar mas

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

¿Dónde colocar el código en la página?

Los códigos JavaScript son insertados a través del <script>. Este elemento tiene un atributo que se utiliza para indicar el tipo de lenguaje que vamos a usar.

JavaScript "en la página"

Para situar el código JavaScript directamente en una página web, siguiendo el ejemplo de ¡Hola, mundo!: se coloca el código en el elemento <script>:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
      <meta content="width=device-width, initial-scale=1" name="viewport">
        <title>¡Hola Mundo!</title>
      </meta>
    </meta>
  </head>
  <body>
      <script>
            alert('¡Hola Mundo!');
      </script>
  </body>
</html>
```

Es importante seguir buenas prácticas al desarrollar un proyecto. Una de ellas es mantener los archivos del proyecto separados. Por ejemplo, si tenemos código

HTML, JavaScript y CSS en un mismo archivo, es necesario separarlos en archivos independientes y llamarlos en el HTML.

JavaScript externo

Es posible, y beneficioso escribir el código JavaScript en un archivo externo con la extensión .js. Este archivo se llama desde la página web mediante el elemento <script> y su atributo src que contiene la dirección URL del archivo .js.

Contenido de ficheros hola.js

```
alert('¡Hola Mundo!');
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
      <meta content="width=device-width, initial-scale=1" name="viewport">
        <title>¡Hola Mundo!</title>
      </meta>
    </meta>
  </head>
  <body>
      <script src="hola.js"></script>
  </body>
</html>
```

El archivo hola.js se encuentra en el mismo directorio que el programa en HTML.

Posición del elemento <script>

Una página web es leída por el navegador de forma lineal, primero lee <head>, después los elementos de <body> uno después del otro. Si se llama a un archivo

JavaScript desde el inicio de la carga de dicha página, el navegador cargará este archivo, y si es robusto, la carga de la página se ralentizará. Esto es normal, ya que el navegador cargará el archivo antes de empezar a mostrar el contenido de la página.

Para resolver este problema, es conveniente colocar los elementos <script> justo antes de cerrar <body> (algunos navegadores lo hacen automáticamente) como el siguiente ejemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
      <meta content="width=device-width, initial-scale=1" name="viewport">
        <title>¡Hola Mundo!</title>
      </meta>
    </meta>
  </head>
  <body>
                <p>
                <!--
                Contenido de la página web como comentario
                ...
                -->
                </p>
                <script>
                // Un poco de código JavaScript...
                </script>
                <script src="hola.js"></script>
  </body>
</html>
```

ISTPET
INSTITUTO TRAVERSARI

CAPÍTULO 2

Variables en JavaScript

CAPÍTULO 2

2.1. Variables en JavaScript

Variables

Una variable es útil para almacenar valores que se utilizan repetidamente en un programa. Los valores que pueden almacenarse dentro de una variable dependiendo de su tipo. Los tipos de datos más comunes son enteros, flotantes, cadenas y booleanos.

Declarar una variable

Una variable se declara con la finalidad de comenzar a guardar datos sin problema dentro de un programa.

Para declarar una variable, primero se debe elegir un nombre. El nombre de la variable puede contener letras, números, guiones bajos etc. El nombre no puede comenzar con un número y no puede ser una palabra reservada.

Las palabras reservadas, tienen un significado especial en JavaScript. No pueden ser usadas como nombres de variables.

Aquí se muestra un ejemplo de cómo declarar una variable:

```
var ejemploVariable;
```

JavaScript es un lenguaje sensible con las mayúsculas y minúsculas. Esto significa que las variables se diferencian por sus mayúsculas y minúsculas. Por ejemplo:

```
var ejemploVariable;
var ejemplovariable;
var EJEMPLOVARIABLE;
```

La palabra clave var, se usa para indicar que se declaró una variable. Una vez que se declara, se puede almacenar cualquier dato:

```
var ejemploVariable;
ejemploVariable = 1;
```

El signo = es utilizado para asignarle un valor a la variable, en el ejemplo se asignó el número 1. Cuando el valor de una variable tiene una asignación, es posible simplificar el código en una sola línea:

```
var ejemploVariable = 2.2 //Como se puede ver, los números decimales se
separan con un punto.
```

2.2. Ámbito de las variables en JavaScript

El alcance de una variable

El alcance de una variable es el lugar del código donde se puede acceder a la variable. En JavaScript, existen dos tipos de alcance: global y local.

Alcance global

Las variables de alcance global están disponibles en todo el código, incluidas las funciones. Se declaran fuera de cualquier función, para declarar una variable global a la página simplemente se hace un script, con la palabra *var*. Como se muestra a continuación en el ejemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
      <meta content="width=device-width, initial-scale=1" name="viewport">
        <title>¡Hola Mundo!</title>
      </meta>
    </meta>
  </head>
  <body>
      <script>
            var variableGlobal
```

```
        </script>
    </body>
</html>
```

Alcance local

Las variables con alcance local solo están disponibles dentro de la función en la que se declara. Cuando se declaran variables locales sólo se puede acceder a ellas donde se ha declarado la variable, es decir, si es declarada en una función solo se puede acceder a ella cuando está dentro de esa función.

Las variables pueden ser locales a una función, pero también pueden ser locales a otros ámbitos, como por ejemplo puede ser un bucle. En general, son ámbitos locales cualquier lugar delimitado por llaves. Ejemplo:

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8">
         <meta content="width=device-width, initial-scale=1" name="viewport">
            <title>¡Hola Mundo!</title>
         </meta>
      </meta>
   </head>
   <body>
         <script>
                        var numero = 1
                          function miFuncion (){
                        var numero = 2
                          document.write(numero) //imprime 2
                        }
                          document.write(numero) //imprime 1
         </script>
```

```
  </body>
</html>
```

Declaración de variables con let o const

Se declaran dentro de una función usando la palabra clave let o const. La sintaxis es la misma que var cuando se va a declarar las variables, pero en este caso let afecta la declaración del bloque.

Bloque significa cualquier espacio qué contiene la acotación de llaves, como podría ser las sentencias que hay dentro de las llaves de un bucle for. Ejemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
      <meta content="width=device-width, initial-scale=1" name="viewport">
        <title>¡Hola Mundo!</title>
      </meta>
    </meta>
  </head>
  <body>
      <script>
            for(let i=0; i<3; i++) {
            // en este caso, la variable i sólo existe dentro del bucle for
                  alert(i);
            }
                  // fuera del bloque for no existe la variable i
      </script>
  </body>
</html>
```

Por otro lado, otra forma de crear variables es "const". Crea una variable donde su valor es inalterable a lo largo del código. Por ejemplo:

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8">
         <meta content="width=device-width, initial-scale=1" name="viewport">
            <title>¡Hola Mundo!</title>
         </meta>
      </meta>
   </head>
   <body>
         <script>
                  const EjVariable = 1;
                           console. log(EjVariable);
                  EjVariable = 10;
                           console.log(EjVariable);
         </script>
   </body>
</html>
```

2.3. Almacenamiento de datos en variables

Los tipos de datos en JavaScript

JavaScript tiene tres tipos de datos principales: numérico, cadena y booleano.

Números

Los números pueden ser enteros, decimales, negativos o en notación científica. Pueden ser asignados a variables de la siguiente forma:

```
var numero = 1;
var numero = 1.1;
var numero = -1;
```

```
var numero = 2.34e4;
var numero = 0x391;
```

Cadenas

Las cadenas incorporan texto. Se puede establecer en variables de dos maneras:

```
var cadena1 = "Mi primera cadena";
var cadena2 = 'Mi segunda cadena';
```

Es relevante tener en cuenta que, si se asigna un número a una variable de tipo cadena, el valor de esta variable sería una cadena en representación del número. Por ejemplo, var ejemplo = "1 " asignará el valor "1 " a la variable ejemplo.

Además, es importante tener en cuenta que, si usa apóstrofos para encerrar una cadena y desea usar apóstrofos en la misma cadena, se debe escapar los apóstrofos. Esto se hace usando una barra invertida (backslash) antes del apóstrofe. Por ejemplo, var ejemplo = 'Esto \' es un ejemplo '; asignará el valor "Esto ' es un ejemplo " a la variable cadena.

Además de declarar cadenas se las puede concatenar, es una técnica útil para combinar dos o mas cadenas de texto se lo hace implementando el operador “+” a continuación vamos a hacer un ejemplo:

```
let nombre = "Juan";
let apellido = "Pérez";
let edad = 25;

let mensaje = "Hola, mi nombre es " + nombre + " " + apellido + " y tengo " + edad + "
años.";
console.log(mensaje);
```

Como resultado saldrá “Hola, mi nombre es Juan Pérez y tengo 25 años”

Booleanos

Los booleanos pueden tener asignados dos valores: verdadero o falso. Se puede establecer en variables de la siguiente manera:

var verdadero = true;

var falso = false;

2.4. Tipos de datos en JavaScript

JavaScript dispone de los siguientes tipos de datos principales:

Tipo de dato	Descripción	Ejemplo básico
Number	Valor numérico (enteros, decimales, etc.)	42
BigInt	Valor numérico grande.	1234567890123456789
String	Valor de texto (cadenas de texto, caracteres, etc.)	'MZ'
Boolean	Valor booleano (valores verdadero o falso)	true
undefined	Valor sin definir (variable sin inicializar)	undefined
Function	Función (función guardada en una variable)	function() {}
Symbol	Símbolo (valor único)	Symbol(1)
Object	Objeto (estructura más compleja)	{}

ISTPET
INSTITUTO TRAVERSARI

CAPÍTULO 3

Operadores

CAPÍTULO 3

Operadores

3.1. Operadores en JavaScript

Los operadores en JavaScript son símbolos que se utilizan para realizar ciertas operaciones en el código. Hay diversos tipos de operadores, cada uno con un propósito específico. Los operadores pueden utilizarse para realizar operaciones matemáticas con los valores de las variables, comparar diferentes variables y realizar cálculos complejos. Por ende, los operadores son una herramienta esencial para cualquier persona interesada en programación. Permiten a los programas tomar decisiones lógicas en función de cotejamiento y otros tipos de condiciones.

Asignación

El operador de asignación se utiliza para guardar un valor en una variable. El símbolo de asignación es "=" . A la izquierda del operador se indica el nombre de la variable, y a la derecha se indica el valor que se va a guardar.
Como, por ejemplo, la siguiente expresión asignaría el valor 3 a la variable numero1:

```
var numero1 = 3;
```

Conjuntamente, se puede asignar el valor de otra variable a una variable. Por ejemplo, la siguiente expresión asignaría el valor de la variable numero2 a la variable numero1:

```
var numero1 = numero2;
```

El operador de asignación sirve para asignar resultados de una operación a una variable. Por ejemplo, la siguiente expresión asignaría el resultado de la suma de 1 y 2 a la variable numero1:

El operador de asignación se puede utilizar para establecer el resultado de una operación a una variable. Por ejemplo, la siguiente expresión estipularía el resultado de la suma de 1 y 2 a la variable numero1:

```
var numero1 = 1 + 2;
```

El operador de asignación es una herramienta fundamental en la programación, porque se utiliza para almacenar valores en las variables.

Incremento y decremento

Los operadores de incremento y decremento son unarios ya que modifican el valor de una variable numérica. El operador de incremento aumenta el valor de la variable en 1, mientras que el operador de decremento lo disminuye en 1.

Los operadores de incremento y decremento se utilizan como prefijo o como sufijo. El prefijo se ubica antes del nombre de la variable, mientras que el sufijo se sitúa después.

Prefijo

El operador de incremento como prefijo se evalúa antes de que se utilice el valor de la variable. Por ejemplo, la siguiente expresión asignará el valor 6 a la variable numero3:

```
var numero1 = 5;
var numero2 = 2;
  numero3 = ++numero1 + numero2;
```

En este caso, el valor de numero1 se incrementa a 6 antes de que se calcule la expresión.

El operador de decremento como prefijo funciona de forma similar. Por ejemplo, la siguiente expresión asignará el valor 4 a la variable numero3:

```
var numero1 = 5;
var numero2 = 2;
  numero3 = --numero1 + numero2;
```

En este caso, el valor de numero1 se disminuye a 4 antes de que se calcule la expresión.

Sufijo

El operador de incremento como sufijo se evalúa después de que se utilice el valor de la variable. Por ejemplo, la siguiente expresión asignará el valor 7 a la variable numero3:

```
var numero1 = 5;
```

```
var numero2 = 2;
numero3 = numero1++ + numero2;
```

En este caso, el valor de numero1 se incrementará a 6 después de que se calcule la expresión.

El operador de decremento como sufijo funciona de forma similar. Por ejemplo, la siguiente expresión asignará el valor 6 a la variable numero3:

```
var numero1 = 5;
var numero2 = 2;
numero3 = numero1-- + numero2;
```

En este caso, el valor de numero1 se disminuirá a 4 después de que se calcule la expresión.

Se recomienda utilizar el operador de incremento como prefijo para evitar errores.

El operador de incremento, cuando se maneja como prefijo, aumenta el valor de la variable antes de que esta se ejecute en la operación. Cuando se utiliza como sufijo, aumenta el valor de la variable después de que se ejecute la operación.

Ejemplos:

```
var numero1 = 5;
var numero2 = 2;

// El valor de numero1 se incrementa después de la suma
numero3 = numero1++ + numero2; // numero3 = 7, numero1 = 6

// El valor de numero1 se incrementa antes de la suma
numero3 = ++numero1 + numero2; // numero3 = 8, numero1 = 6
```

Lógicos

Los operadores lógicos son esenciales, ya que permiten tomar decisiones en función de condiciones.

Los operadores lógicos son utilizados para combinar dos o más expresiones lógicas, estas pueden ser verdaderas o falsas. El resultado de cualquier operación con operadores lógicos siempre es un valor lógico, que puede ser verdadero (true) o falso (false).

Los operadores lógicos más comunes son:

- **AND (&&):** Devuelve true si ambas expresiones son verdaderas.
- **OR (||):** Devuelve true si al menos una de las expresiones es verdadera.
- **NOT (!):** Devuelve el valor opuesto de la expresión.

Por ejemplo, la siguiente expresión devolverá true si el valor de la variable edad es mayor de 18 y menor de 25:
(edad > 18) && (edad < 25)

La siguiente expresión devolverá true si el valor de la variable edad es mayor o igual a 18 o menor o igual a 25:
(edad >= 18) || (edad <= 25)

La siguiente expresión devolverá true si el valor de la variable edad no es mayor de 30:
!(edad > 30)

Los operadores lógicos se utilizan en una amplia variedad de contextos, como:

- **Condicionales:** Para determinar si se debe ejecutar o no una instrucción.
- **Bucles:** Para controlar el número de veces que se ejecuta una instrucción.
- **Funciones:** Para determinar si una función debe retener un valor.

Relacionales

Los operadores relacionales de JavaScript son similares a los operadores relacionales de matemáticas. Permiten comparar dos valores y devolver un valor booleano: verdadero o falso.

Los operadores relacionales más comunes son:

- **Mayor que (>):** Devuelve verdadero si el primer valor es mayor que el segundo.
- **Menor que (<):** Devuelve verdadero si el primer valor es menor que el segundo.
- **Mayor o igual que (>=):** Devuelve verdadero si el primer valor es mayor o igual que el segundo.
- **Menor o igual que (<=):** Devuelve verdadero si el primer valor es menor o igual que el segundo.
- **Igual que (==):** Devuelve verdadero si ambos valores son iguales.
- **Distinto de (!=):** Devuelve verdadero si ambos valores son diferentes.

Los operadores relacionales se manejan en una amplia variedad de contextos, como:

- **Condicionales:** Para determinar si se debe ejecutar o no una instrucción.
- **Bucles:** Para controlar el número de veces que se ejecuta una instrucción.
- **Funciones:** Para determinar si una función debe devolver o no un valor.

Es importante tener cuidado con el operador de igualdad (==), ya que puede causar errores de programación. El operador == se utiliza para comparar el valor de dos variables, por lo que es muy diferente del operador =, que se utiliza para asignar un valor a una variable.

Ejemplos:

```
var numero1 = 3;
var numero2 = 5;

*/ El operador > devuelve verdadero si el primer valor es mayor que el segundo*/
resultado = numero1 > numero2; // resultado = false

// El operador < devuelve verdadero si el primer valor es menor que el segundo
```

```
resultado = numero1 < numero2; // resultado = true

// El operador >= devuelve verdadero si el primer valor es mayor o igual que el segundo
resultado = numero1 >= numero2; // resultado = false

// El operador <= devuelve verdadero si el primer valor es menor o igual que el segundo
resultado = numero1 <= numero2; // resultado = true

// El operador == devuelve verdadero si los dos valores son iguales
resultado = numero1 == numero2; // resultado = false

// El operador! = devuelve verdadero si los dos valores son diferentes
resultado = numero1 != numero2; // resultado = true
```

Operadores relacionales con cadenas de texto:

Los operadores relacionales, se utilizan con variables de tipo cadena de texto. Sin embargo, su funcionamiento es diferente al de los operadores relacionales con números.

Cuando se utilizan cadenas de texto, los operadores "mayor que" (>) y "menor que" (<) comparan letra a letra comenzando desde la izquierda hasta que se encuentre una diferencia entre las dos cadenas de texto. Si las cadenas de texto son iguales, los operadores devuelven verdadero. Si las cadenas de texto son diferentes, los operadores devuelven falso.

Ejemplo:

```
var texto1 = "hola";
var texto2 = "hola";
var texto3 = "adios";

// El operador == devuelve verdadero si las cadenas de texto son iguales
resultado = texto1 == texto2; // resultado = true
```

```
// El operador != devuelve verdadero si las cadenas de texto son diferentes
resultado = texto1 != texto3; // resultado = true

// El operador >= devuelve verdadero si la primera cadena de texto es mayor o igual que la segunda
resultado = texto3 >= texto2; // resultado = false
```

Recomendaciones:

- Es importante utilizar los operadores relacionales correctamente para evitar errores de programación.
- Es importante tener cuidado con el operador de igualdad (==), ya que puede causar errores de programación.
- Es importante tener en cuenta que el funcionamiento de los operadores relacionales con cadenas de texto es diferente al de los operadores relacionales con números.

3.2. Operador ternario en JavaScript

El operador condicional, también conocido como operador ternario, es un operador de tres operandos y se utiliza para realizar una evaluación condicional.

Sintaxis

El operador condicional tiene la siguiente sintaxis:

```
condición? expr1 : expr2
```

Parámetros

- **condición:** Una expresión que se evalúa como verdadero o falso.
- **expr1:** La expresión que se devuelve si la condición es verdadera.
- **expr2**: La expresión que se devuelve si la condición es falsa.

Descripción

Si la condición es verdadera, el operador condicional devuelve el valor de la expresión expr1. De lo contrario, devuelve el valor de la expresión expr2.

Ejemplos

```
// Muestra un mensaje diferente en función del valor de la variable `esMiembro`.

var esMiembro = true;

console.log ("La cuota es de: " + (esMiembro? "$100”: "$200"));
// Salida: "La cuota es de: $100"

// Asigna una variable en función del resultado de la condición ternaria.

var materia = Matematicas.PI > 4 ? "Sí" : "No";
// `materia` se asigna a "No"
// Realiza evaluaciones ternarias múltiples.

var firstCheck = false,
  secondCheck = false,
  acceso;

acceso = PrimerIngreso
  ? "Acceso denegado"
  : SegundoIngreso
  ? "Acceso denegado"
  : "Acceso permitido";

console.log(access); // Salida: "Acceso permitido"

// Usa operaciones ternarias en espacio vacío.
```

```
var preguntar = false,
edad = 16;
edad > 18 ? location.assign("continue.html") : (stop = true);
// `stop` se asigna a `true`

// Realiza más de una operación por caso, separándolas con una coma.

var preguntar = false,
edad = 23;
edad > 18
  ? (alert("OK, puedes continuar."), location.assign("continue.html"))
  : ((stop = true), alert("Disculpa, eres menor de edad!"));

// Realiza más de una operación durante la asignación de un valor.
var edad = 16;
var url =
edad > 18
    ? (alert("OK, puedes continuar."),
      // `alert()` devuelve "undefined", pero será ignorado porque
      // no es el último valor separado por comas del paréntesis
      "continue.html")
    : (alert("Eres menor de edad!"),
      alert("Disculpa :-("),
      // etc. etc.
      "stop.html");
location.assign(url); // "stop.html"
```

3.4. Operador typeof de JavaScript para control de tipos

El operador typeof, devuelve el tipo de dato de una variable. Esto es ventajoso porque JavaScript es un lenguaje dinámico, lo que significa que las variables pueden cambiar de tipo durante la ejecución de un programa.

El operador typeof tiene la siguiente sintaxis:

El operador typeof devuelve una cadena de texto que representa el tipo de dato de la variable. Los tipos de datos que devuelve el operador typeof son:

- **number:** Números
- **string:** Cadenas de texto
- **boolean:** Valores booleanos (true o false)
- **undefined:** Variables que no han sido definidas
- **object:** Objetos
- **function:** Funciones
- **array:** Arreglos
- **symbol:** Símbolos
-

Por ejemplo:

```
var x = 12345;
console.log(typeof x); // number

var y = 'Hola, mundo!';
console.log(typeof y); // string

var z = true;
console.log(typeof z); // boolean

var a;
console.log(typeof a); // undefined
var b = {};
console.log(typeof b); // object

var c = function() {
  // ...
};
```

```
console.log(typeof c); // function

var d = [1, 2, 3, 4];
console.log(typeof d); // array

var e = Symbol('miSimbolo');
console.log(typeof e); // symbol
```

El operador typeof, se puede utilizar para comprobar si una variable ha sido definida. Si la variable no ha sido definida, typeof devolverá el valor undefined.

Por ejemplo:

```
var x;
console.log(typeof x); // undefined
```

Es importante tener en cuenta que el operador typeof no siempre devuelve el valor que esperamos. Por ejemplo, los números NaN (no es un número) son tratados como números por el operador typeof.

Por ejemplo:

```
var x = NaN;
console.log(typeof x); // número
```

Para comprobar si un valor es un número, podemos utilizar la función Numero.esNaN().

Por ejemplo:

```
function esNumero(value) {
  return !Numero.esNaN(value);
}
console.log(esNumero(NaN)); // false
```

Se debe tener en cuenta que algunos valores en JavaScript son inesperadamente objetos. Por ejemplo, los arreglos y los valores null son tratados como objetos por el operador typeof.
Por ejemplo:

```
var x = [1, 2, 3];
console.log(typeof x); // object
var y = null;
console.log(typeof y); // object
```

3.5. Ejercicios

1. Para entender mejor operadores y su precedencia se debe establecer el orden en el que se evalúan las operaciones. Los operadores con mayor precedencia se evalúan primero, seguidos de los operadores con menor precedencia. La precedencia de los operadores se pondrá en práctica en el siguiente ejercicio:

- La multiplicación tiene mayor precedencia que la suma, por lo que primero se multiplica 5 y 3, y luego se agrega 10.

```
var c = 5 * 3 + 10;

console.log(c);
```

- Los operadores tienen la misma precedencia, por lo que se evalúan de izquierda a derecha.

```
var d = 5 + 3 * 10;

console.log(d);
```

- Los operadores de comparación tienen mayor precedencia que los operadores aritméticos, por lo que primero se realiza la comparación y luego se realiza la resta.

```
var e = 10 > 5 - 3;
```

console.log(e);

- En este caso, se utilizan paréntesis para cambiar el orden de evaluación.

var f = (10 > 5) - 3;

console.log(f);

La salida de este código es la siguiente:

25

35

false

7

Ingrese el código en el editor de texto de su preferencia, como se muestra a continuación:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
      <meta content="width=device-width, initial-scale=1" name="viewport">
        <title>¡Hola Mundo!</title>
      </meta>
    </meta>
  </head>
  <body>
      <script>
            var a = 10;
            var b = 5;
            var c = 5 * 3 + 10;
                  console.log(c); // 25
            var d = 5 + 3 * 10;
```

```
            console.log(d); // 35
        var e = 10 > 5 - 3;
            console.log(e); // false
        var f = (10 > 5) - 3;
            console.log(f); // 7
        </script>
    </body>
</html>
```

ISTPET
INSTITUTO TRAVERSARI

CAPÍTULO 4

Estructuras de control en JavaScript

CAPÍTULO 4

4.1. Estructuras de control en JavaScript

Los programas que utilizan variables y operadores son un encadenamiento lineal de instrucciones que se ejecutan una tras otra. Sin embargo, estos programas no pueden tomar decisiones en función del valor de las variables ni repetir la misma instrucción un número determinado de veces.

Sin las estructuras de control, los programas serían una simple sucesión lineal de instrucciones.

- **Semántica:** Se utiliza un lenguaje más simple y directo para que el texto sea más fácil de entender. Se sustituye "estructuras de control de flujo" por "instrucciones que permiten al programa tomar decisiones o repetir una instrucción".
- **Claridad:** Sirve para ilustrar cada tipo de estructura de control de flujo.

- **Concisión:** Se eliminan las redundancias y la información innecesaria. La ejecución condicional es una herramienta que permite a los programas tomar decisiones en función del valor de las variables.

4.2. Estructura IF en JavaScript

Los operadores en JavaScript son símbolos que se utilizan para realizar ciertas operaciones en el código. Hay diversos tipos de operadores, cada uno con un propósito específico. Los operadores pueden utilizarse para realizar operaciones matemáticas con los valores de las variables, comparar diferentes variables y realizar cálculos complejos. Por ende, los operadores son una herramienta esencial para cualquier persona interesada en programación. Permiten a los programas tomar decisiones lógicas en función de cotejamiento y otros tipos de condiciones.

Ejemplo:

El siguiente código muestra cómo utilizar la ejecución condicional para determinar si un número es par o impar.

```
var numero = prompt("Elige un número");
if (numero % 2 === 0) {
  console.log("El número es par");
} else {
  console.log("El número es impar");
}
```

En este ejemplo, el bloque de código entre llaves se ejecutará si el número es par. Si el número es impar, se ejecutará el bloque de código de la rama else.

4.3. Estructura SWITCH de JavaScript

En JavaScript, la estructura de control switch se utiliza para tomar decisiones en función del valor de una variable. Las estructuras de control son instrucciones que permiten al programa tomar decisiones o repetir una instrucción un número determinado de veces, la estructura switch es una estructura de control más compleja que permite al programa tomar múltiples decisiones en función del valor de una variable se puede comparar el valor de una variable con varios casos y ejecutar diferentes bloques de código en función del caso que coincida.

Ejemplo:

El siguiente código muestra cómo utilizar la estructura switch para determinar el día de la semana en función de un número:

```
let dia = prompt("¿Qué día es hoy?");

switch (dia) {
```

```
  case "lunes":
    console.log("Hoy es lunes");
    break;
  case "martes":
    console.log("Hoy es martes");
    break;
  case "miércoles":
    console.log("Hoy es miércoles");
    break;
  case "jueves":
    console.log("Hoy es jueves");
    break;
  case "viernes":
    console.log("Hoy es viernes");
    break;
  case "sábado":
    console.log("Hoy es sábado");
    break;
  case "domingo":
    console.log("Hoy es domingo");
    break;
  default:
    console.log("No se reconoce el día");
    break;
}
```

En este ejemplo, el programa comprueba si el valor de la variable dia concuerda con uno de los valores indicados en los casos de la estructura switch. Si lo hace, se ejecutará el código correspondiente a ese caso. Si no lo hace, se ejecutará el código del caso default.

- La estructura switch permite al programa ejecutar código en función del valor de una variable.
- El código se ejecutará en la etiqueta de case que coincida con el valor de la variable.
- Si no se encuentra ninguna coincidencia, el código se ejecutará en la etiqueta de case default.
- El código continuará ejecutándose hasta que se encuentre una declaración break.
- En algunos casos, se puede utilizar una declaración break para compartir código entre casos.

Sin embargo, es importante tener cuidado, ya que es fácil olvidarse de incluir una declaración break.

4.4. Bucle FOR en JavaScript

- Los ciclos for son una forma más concisa de escribir ciclos while.
- Los ciclos for tienen tres partes: la inicialización, la condición de continuación y la actualización.
- La inicialización se ejecuta una vez, antes de que comience el ciclo.
- La condición de continuación se evalúa al comienzo de cada iteración del ciclo. Si la condición es verdadera, el ciclo continúa. Si la condición es falsa, el ciclo termina.
- La actualización se ejecuta al final de cada iteración del ciclo.

El siguiente ejemplo muestra cómo utilizar un ciclo for para imprimir los números pares del 0 al 10:

```
for (var numero = 0; numero <= 10; numero = numero + 2) {
  console.log(numero);
}
```

El siguiente ejemplo muestra cómo utilizar un ciclo for para calcular 210:

```
var resultado = 1;
```

```
for (var contador = 0; contador < 10; contador = contador + 1) {
  resultado = resultado * 2;
}
console.log(resultado);
```

Los ciclos pueden terminar de varias maneras

La forma más común es que la condición del ciclo se evalúe como falsa.

Otra forma de terminar un ciclo es utilizar la declaración break.

La declaración break sale inmediatamente del ciclo circundante.

El siguiente ejemplo muestra cómo utilizar la declaración break para encontrar el primer número mayor o igual a 20 que es divisible por 7:

```
for (var actual = 20; ; actual = actual + 1) {
  if (actual % 7 == 0) {
    console.log(actual);
    break;
  }
}
```

4.5. Bucles WHILE y DO WHILE

El bucle WHILE es una estructura de control que se utiliza para repetir la ejecución de un conjunto de instrucciones mientras se cumpla una condición. Es más sencillo de entender que el bucle FOR, ya que no requiere inicializar, evaluar y actualizar una variable de control en la misma línea. Solo se indica la condición que debe cumplirse para que se realice una iteración.

Un programa que muestra todos los números pares de 0 a 10 puede escribirse de la siguiente manera:

```
console.log(0);
console.log(2);
console.log(4);
```

```
console.log(6);
console.log(8);
console.log(10);
```

La idea de escribir un programa es hacer algo más eficiente, no más trabajo. Por ejemplo, si necesitamos todos los números pares menores a 1000, escribir un número por línea sería poco práctico, ya que tendríamos que escribir 500 líneas de código.

Lo que necesitamos es una forma de ejecutar una pieza de código varias veces. Esta forma de flujo de control se llama bucle. Un bucle nos permite regresar a algún punto del programa y repetirlo.

Por ejemplo, podemos utilizar un bucle para imprimir todos los números pares menores a 1000 de la siguiente manera:

```
for (var i = 0; i < 1000; i++) {
  if (i % 2 === 0) {
    console.log(i);
  }
}
```

Este bucle se ejecutará 500 veces, una vez por cada número par menor a 1000. En cada iteración, el bucle comprobará si el número actual es par. Si lo es, lo imprimirá.

Esta es una forma mucho más eficiente de imprimir todos los números pares menores a 1000 que escribir un número por línea.

ISTPET
INSTITUTO TRAVERSARI

CAPÍTULO 5

¿Dónde colocamos las funciones JavaScript?

CAPÍTULO 5

5.1. ¿Dónde colocamos las funciones JavaScript?

Los programas robustos son más propensos a tener errores a diferencia de los programas pequeños. Esto se debe a que la complejidad de los programas grandes puede dificultar que los programadores los entiendan.

Si consideramos solo el tamaño del código, el segundo programa es más grande que el primero. Sin embargo, el segundo programa es más probable que sea correcto porque se expresa en un vocabulario que corresponde al problema que se está resolviendo.

Para reducir el riesgo de errores, los programadores deben tratar de mantener sus programas lo más simples posible.

Abstraer la repetición

Las funciones simples son una buena forma de construir abstracciones, pero a veces no son suficientes.

Es común que un programa haga algo una determinada cantidad de veces. Por ejemplo, podemos escribir un ciclo for para imprimir los números del 1 al 10:

JavaScript

```
for (let i = 1; i <= 10; i++) {
  console.log(i);
}
```

¿Podemos abstraer la acción de "hacer algo N veces" como una función?

Sí, podemos. Podemos escribir una función que llame a otra función N veces. Por ejemplo, la siguiente función llamará a la función console.log() 10 veces:

```
function repetir(n, accion) {
  for (let i = 0; i < n; i++) {
    accion(i);
  }
}
```

```
repetir(10, console.log);
```

Esta función está estructurada de manera similar a un ciclo for: primero describe el tipo de ciclo y luego proporciona un cuerpo. Sin embargo, el cuerpo ahora está escrito como una función, que está envuelta en los paréntesis de la llamada a repetir.

En casos como este, donde el cuerpo es una expresión pequeña y única, podemos omitir las llaves y escribir el ciclo en una sola línea:

```
repetir(10, i => console.log(i));
```

Funciones de orden superior

Las funciones que operan en otras funciones se denominan funciones de orden superior.

Las funciones de orden superior nos permiten abstraer sobre acciones, no solo sobre valores. Por ejemplo, podemos tener funciones que crean nuevas funciones:

```
function mayorQue(n) {
  return m => m > n;
}

let mayorQue10 = mayorQue(10);
console.log(mayorQue10(11)); // → true
```

También podemos tener funciones que cambian otras funciones:

```
function ejemplo(funcion) {
  return (...argumentos) => {
    console.log("Llamado con", argumentos);
    let resultado = funcion(...argumentos);
    console.log("Llamado con", argumentos, ", retorno", resultado);
    return resultado;
  };
}
```

```
ejemplo (Math.min)(3, 2, 1);
// → Llamado con [3, 2, 1]
// → 1
// → Llamado con [3, 2, 1], retorno 1
```

5.2. Parámetros de las funciones

Parámetros

Los parámetros son variables que se pasan a una función desde el exterior. Se utilizan para proporcionar información a la función o para modificar su comportamiento.

Ejemplo:

En el siguiente ejemplo, la función tablaDelUno() recibe un parámetro llamado "hasta". Este parámetro indica hasta qué número debe llegar la tabla de multiplicar del uno.

```
function tablaDelUno(hasta) {
  for (let i = 0; i <= hasta; i++) {
    console.log("1 x", i, "=", 1 * i);
  }
}
```

Ejecución

La función tablaDelUno() se puede ejecutar con cualquier valor para el parámetro "hasta". En el siguiente ejemplo, la función se ejecuta dos veces: una vez con hasta = 10 y otra vez con hasta = 5.

```
tablaDelUno(10); // Tabla del 1
tablaDelUno(5);  // Tabla del 1
```

Análisis

Cuando se ejecuta la función tablaDelUno(10), el parámetro hasta tiene el valor 10. Esto significa que el bucle for se ejecutará 11 veces, desde i = 0 hasta i = 10. Como resultado, se mostrará la tabla de multiplicar del uno, desde el 0 hasta el 10.

Cuando se ejecuta la función tablaDelUno(5), el parámetro hasta tiene el valor 5. Esto significa que el bucle for se ejecutará 6 veces, desde i = 0 hasta i = 5. Como resultado, se mostrará la tabla de multiplicar del uno, desde el 0 hasta el 5.

Parámetros múltiples

Las funciones pueden recibir varios parámetros. En este caso, los parámetros se separan por comas.

Parámetros por defecto

Los parámetros pueden tener valores predeterminados. Esto significa que, si no se especifica un valor para el parámetro al llamar a la función, se utilizará el valor predeterminado.

Devolución de valores

Las funciones pueden devolver valores. Para ello, se utiliza la palabra clave return.

Paráfrasis

Las funciones en JavaScript pueden recibir información desde el exterior a través de parámetros. Los parámetros son variables que se pasan a la función cuando se llama. Los parámetros se pueden utilizar para proporcionar información a la función o para modificar su comportamiento.

Las funciones pueden recibir varios parámetros. En este caso, los parámetros se separan por comas.

Los parámetros pueden tener valores predeterminados. Esto significa que, si no se especifica un valor para el parámetro al llamar a la función, se utilizará el valor predeterminado.

Las funciones también pueden devolver valores. Para ello, se utiliza la palabra clave return.

Ejemplos:

JavaScript

```
// Función con un parámetro
function saludar(nombre) {
  console.log("Hola, " + nombre);
}

// Llamada a la función con un parámetro
saludar("Juan"); // Imprime "Hola, Juan"

// Función con dos parámetros
function sumar(a, b) {
  return a + b;
}

// Llamada a la función con dos parámetros
const resultado = sumar(5, 5); // Resultado = 10

// Función con un parámetro por defecto
function sumar(a = 1, b) {
  return a + b;
}

// Llamada a la función con un parámetro
const resultado = sumar(5); // Resultado = 6

// Llamada a la función sin especificar el parámetro
const resultado = sumar(); // Resultado = 2
```

Los parámetros son una característica importante de las funciones en JavaScript. Los parámetros permiten que las funciones sean más flexibles y reutilizables.

5.3. Valores de retorno en funciones JavaScript

Valores de retorno

Las funciones en JavaScript pueden producir valores, además de efectos secundarios.

Efectos secundarios

Los efectos secundarios son acciones que una función realiza en el mundo exterior, como mostrar un cuadro de diálogo o escribir texto en la pantalla.

Valores

Los valores son datos que una función puede devolver.

Ejemplo:

La función Math.max() toma cualquier cantidad de argumentos numéricos y devuelve el mayor de ellos.

```
console.log(Math.max(2, 4));
```

Lo que va a devolver es 4.

Palabra clave return

La palabra clave return se utiliza para devolver un valor desde una función.

Ejemplo:

La siguiente función devuelve la suma de dos números:

```
function sumar(a, b) {
  return a + b;
}
console.log(sumar(2, 4));
```

Lo que va a devolver es 6

Uso de funciones en expresiones

Las funciones se pueden usar en expresiones como cualquier otra variable.

```
console.log(Math.min(2, 4) + 5);
```
Lo que va a devolver es 7.

Las funciones en JavaScript pueden producir valores, además de efectos secundarios. Los valores de retorno permiten que las funciones sean más flexibles y reutilizables.

5.4. Funciones integradas en el lenguaje JavaScript

JavaScript contiene una serie de funciones que no están asociadas a ningún objeto. Estas funciones son útiles para realizar tareas específicas, como evaluar expresiones, convertir cadenas de texto a números o comprobar si un valor es un número.

Función eval

La función eval() evalúa una expresión JavaScript. Esta expresión puede ser una cadena de texto, un objeto o una función.

```
// Ejemplo de uso de la función eval()
var miTexto = "3 + 5";

// Evalua la expresión 3 + 5 y devuelve el resultado, que es 8
var resultado = eval(miTexto);

console.log(resultado); // 8
```

Función parseInt

La función parseInt() convierte una cadena de texto a un número. La función recibe dos parámetros: el texto a convertir y la base en la que está escrito el texto.

```
// Ejemplo de uso de la función parseInt()
var miTexto = "34";
```

```
// Convierte el texto "34" a un número en base 10
var numero = parseInt(miTexto);
console.log(numero); // 34
```

La función parseInt() también puede recibir un segundo parámetro para indicar la base en la que está escrito el texto. Los valores posibles para la base son 2, 8, 10 y 16.

```
// Ejemplo de uso de la función parseInt() con una base
var miTexto = "101011";
// Convierte el texto "101011" a un número en base 2
var numero = parseInt(miTexto, 2);
console.log(numero); // 43
```

Función isNaN

La función isNaN() comprueba si un valor es un número. La función recibe un parámetro, que es el valor a comprobar.

```
// Ejemplo de uso de la función isNaN()

var miTexto = "34";

// Comprueba si el texto "34" es un número
var esNumero = !isNaN(miTexto);

console.log(esNumero); // true
```

Las funciones de JavaScript sin objetos son útiles para realizar tareas específicas. Estas funciones pueden ser utilizadas en combinación con otras funciones para realizar tareas más complejas.

5.5. Función console.log

La función console.log() es una función estándar de JavaScript que se utiliza para imprimir valores en la consola del navegador. La consola es una herramienta de desarrollo que permite a los desarrolladores ver el resultado de sus código.

Uso de la función console.log()

La función console.log() se utiliza para imprimir valores en la consola. Los valores pueden ser de cualquier tipo, incluidos números, cadenas, objetos y funciones.
Ejemplos de uso de la función console.log()`

```
// Imprime el número 10 en la consola
console.log(10);

// Imprime la cadena "Hola, mundo" en la consola
console.log("Hola, mundo");

// Imprime el objeto { nombre: "Juan", edad: 25 } en la consola
console.log({ nombre: "Juan", edad: 25 });

// Imprime la función sumar(a, b) en la consola
console.log(function sumar(a, b) {
  return a + b;
});
```

Explicación de la sintaxis de la función console.log()

La función console.log() toma un número variable de argumentos. Cada argumento se imprime en la consola, separados por un espacio.

La función console.log()` no es una vinculación.

La función console.log() no es una vinculación simple. En realidad, es una expresión que obtiene la propiedad log del valor mantenido por la vinculación console.

5.6. Ejercicios

5.6.1 Imprimir mediante consola un arreglo de números y activarlo por un botón

Código:

```
<html>
 <head></head>
 <script>
   function arreglo() {
     var arreglo = [1, 2, 3, 4, 5]; //Creación de un array con un conjunto de datos numéricos con e nombre de variable "arreglo"
     console.log(arreglo); //Se imprime meidante consola el arreglo definido como "arreglo"
   }
 </script>
 <body>
   <input type="button" id="bt" value="MOSTRAR ARREGLO" onclick="arreglo()" />
 </body>
</html>
```

5.6.2 Imprimir el resultado de una operación matemática en la consola

Código:

```
<html>
 <head></head>
 <script>
   function suma() {
       var a = 20;  //Definición de una variable con valor de 20 y nombre a
       var b = 7;  //Definición de una variable con valor de 7 y nombre b
```

```
        console.log("La suma de", a, "y", b, "es:", a + b); // Imprime: La suma de 20 y 7 es:
27
    }
  </script>
  <body>
    <input type="button" id="bt" value="MOSTRAR CÁLCULO" onclick="suma()" />
  </body>
</html>
```

ISTPET
INSTITUTO TRAVERSARI

CAPÍTULO 6

Arrays JavaScript

CAPÍTULO 6

Arrays JavaScript

6.1. Arrays en JavaScript

Un array es una estructura de datos que permite almacenar una colección de elementos en una sola variable. Cada elemento en un array se identifica mediante un índice que representa su posición en el array. En JavaScript, los índices de los arrays comienzan en 0. La longitud y el tipo de los elementos de un array pueden variar esto significa que un array puede contener cualquier tipo de datos, como números, cadenas de texto, objetos, e incluso otros arrays. Además, la cantidad de elementos en un array puede cambiar en cualquier momento.

Es importante tener en cuenta que los arrays de JavaScript no son necesariamente densos. Esto significa que los elementos de un array pueden estar almacenados en ubicaciones no contiguas en la memoria. Por lo tanto, no hay garantía de que los arrays de JavaScript sean densos, lo que depende de cómo el programador elija usarlos.

Un array en JavaScript es una variable que puede almacenar una colección de datos. Los elementos de un array se pueden acceder mediante su índice, que es un número que representa su posición en el array.

```
// Crea un array de tres números
var numbers = [1, 2, 3];

// Accede al segundo elemento del array
var secondNumber = numbers[1];

// Añade un elemento al array
numbers.push(4);

// Elimina el último elemento del array
numbers.pop();
```

6.2. Longitud de los arrays

La longitud de arrays se define como la cantidad que contiene un arreglo en el número de datos es decir todos los datos que se hayan definido dentro del arreglo, en el lenguaje JavaScript se puede obtener la longitud del array con la propiedad “length”.

Ejercicio: *Mostar la longitud de un array*

```
//Muestra de la longitud del arreglo
var array1 = [1, 2, 3, 4, 5];
console.log(array1.length); // La longitud será de 5
```

6.3. Arrays multidimensionales en JavaScript

Se define como array multidimensional a un tipo de array que contiene varios elementos de tipo arreglo y que forman parte de un solo conjunto, en otras palabras, es un array que contiene a otro conjunto de arreglos definidos como uno solo.

Ejercicio: *Creación de un array multidimensional con 3 conjuntos de datos e imprimir en consola los siguientes términos 1 y 6 que se definan en la posición 0,0 y 1,2*

```
var matriz = [
    // Creación de un array multidimensional
    [1, 2, 3], //array1
    [4, 5, 6], //array2
    [7, 8, 9], //array3
];
// Acceder a un elemento en el arreglo denominado matriz

console.log(matriz[0][0]); // Imprime: 1
```

```
console.log(matriz[1][2]); // Imprime: 6
```

6.4. Recorridos forEach sobre Arrays de JavaScript

forEach es una función que esta incorporada en JavaScript donde su principal funcionalidad es recorrer a todos los elementos que se encuentren dentro de un array y dar ejecución de una función por cada elemento en el proceso.

Ejercicio: *Realizar una suma con todos los datos dentro de un array utilizando la propiedad forEach*

```
//Creación del conjunto de datos
var numeros = [1, 2, 3, 4, 5];
// Variable para almacenar la suma
var suma = 0;
// Uso forEach para sumar los números
numeros.forEach(function(numero) {
   suma += numero;
});
// Imprimir el resultado
console.log("La suma de los números es:", suma);
```

6.5. Método map de los arrays en JavaScript

Map es otra función incorporada en JavaScript que comparte relación parcial con forEach pero no con la misma funcionalidad ya que ambos pueden recorrer a todos los datos definidos en un array con la diferencia de forEach, map devuelve un nuevo conjunto de datos en base a los resultados de las funciones que se aplicaron al array original.

Ejercicio: *Crear un array que calcule sus datos por 2 y muestre los nuevos resultados mediante la propiedad map*

```
var numeros = [1, 2, 3, 4, 5]; //Creación de un array
    // Utilizar map para duplicar cada número
    var numerosDuplicados = numeros.map(function (numero) {
      return numero * 2;
    });
    // Imprimir el nuevo array con los números duplicados
    console.log(numerosDuplicados)
```

6.6. Ordenación de arrays en JavaScript con array.sort()

Una propiedad de JavaScript muy útil ya que tiene como funcionalidad ordenar los datos que encuentre en un array lo que significa que modifica al array original para poder ordenarlo.

Ejercicio: *Crear un array con nombres de frutas y ordenarlo mediante la función sort.*

```
// Definir un array con nombres de frutas desordenado
var frutas = ["Manzana", "Banana", "Cereza", "Uva"];
// Uso de sort para ordenar el array
frutas.sort();
// Imprimir el array ordenado alfabéticamente
console.log(frutas)
```

Ejercicio: *Crear un array con datos numéricos y ordenarlos mediante la función sort*

```
// Definir un array de números desordenado
var numeros = [10, 5, 8, 1, 7];
// Utilizar sort con una función de comparación para ordenar numéricamente
numeros.sort(function(a, b) {
   return a - b;
});
// Imprimir el array ordenado numéricamente
console.log(numeros);
```

6.7. Ejercicios

Ejercicio: *Creación de un array multidimensional con 4 conjunto de datos que contengan números del 1 al 12 e imprimir solo los datos impares*.

Código:

```
var matriz = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9],
    [10,11,12]
    ];
console.log(matriz[0][1]);
console.log(matriz[1][0]);
console.log(matriz[1][2]);
```

```
console.log(matriz[2][1]);

console.log(matriz[3][0]);

console.log(matriz[3][2]);
```

Ejercicio: *Crear un array con 10 números que eleve al cubo sus dígitos e imprimirlos en consola.*

Código:

```
var conjunto = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

    var numerosElevados =conjunto.map(function (valor) {

      return valor * 3;

    })

    console.log(numerosElevados)
```

ISTPET
INSTITUTO TRAVERSARI

CAPÍTULO 7

Objetos en JavaScript

CAPÍTULO 7

Objetos en JavaScript

7.1 Introducción general a los objetos en JavaScript

En la programación existe un elemento muy importante que sirve de base fundamental para poder desarrollar de una forma estructurada y son los objetos que son estructuras de datos que agrupan y organizan a un grupo de valores o atributos propios bajo un mismo concepto o encapsulándolos en uno solo una muestra de lo que a valores refiere se pueden agregar métodos a los objetos.

Ejemplo:

```
let persona = {
  nombre: "Gabriel",
  edad: 20,
  profesion: "Desarrollador"
};
```

El objeto llamado “persona” encapsula y guarda 3 atributos informativos como el nombre, edad y profesión haciendo de estos tres datos uno solo.

7.2 Literales de objeto en JavaScript

Los literales de objeto son una forma de crear objetos mejores estructurados o que cuenten con una declaración mucho más directa y concisa, simplemente se define como como una lista de pares clave-valor donde clave se refiere a un atributo declarado en el objeto y cada valor es el dato asociado con dicha propiedad.

Ejemplo:

```
// Definir un objeto llamado persona con varias propiedades propias del objeto
let persona = {
```

```
  nombre: "Elena",
  edad: 30,
  direccion: {
    calle: "123 Calle Principal",
    ciudad: "Ciudad Ejemplo",
    codigoPostal: "12345"
  }
};

// Acceder a las propiedades del objeto
console.log(persona.nombre); // Salida: Elena
console.log(persona.direccion.ciudad); // Salida: Ciudad
```

Ejemplo:

En este ejemplo se realizó la creación de un objeto llamado persona con los atributos de: nombre, edad y se crea un nuevo objeto dentro de persona el cual tiene datos de la dirección. En la impresión de datos se llama al objeto persona y a un atributo del propio objeto, mientras que en la segunda línea de la consola se puede llamar al objeto persona y también al objeto que lo contiene que es dirección con el atributo ciudad

7.3 for in en JavaScript

El uso del bucle for..in en JavaScript es importante a la hora de interactuar con cada propiedad que este declarada en un objeto, esto permite tener una funcionalidad con cada atributo que se cree en el conjunto de datos.

Ejemplo:

```
let persona = {
  nombre: "Juan",
  edad: 25,
  profesion: "Desarrollador"
};
// Interacción sobre las propiedades del objeto con for...in
for (let clave in persona) {
  console.log(`${clave}: ${persona[clave]}`);
}
```

Se hace la creación de un objeto llamado persona con tres atributos nombre, edad y profesión donde mediante el bucle for..in en la variable clave se van asignando los valores de nombre, edad y profesión acorde a los datos declarados en el objeto

7.4 Creación de clases en JavaScript tradicional

En JavaScript, la creación de clases se ha realizado con la ayuda de otros elementos como lo son funciones, constructores o prototipos, aunque en la actualidad se lo realice de una forma diferente gracias a la sintaxis de clases agregada en ECMAScript 2015 (ES6) sin embargo es importante entender el modo de creación ambiguo es una buena práctica ya que ayuda a entender la lógica del mismo concepto. A continuación, se presentará un ejemplo de cómo se realizaban las clases de forma antigua.

Ejemplo 1:

```
//Definir una función en forma de constructor
function Persona(nombre, edad, profesion) {
```

```
  this.nombre = nombre;
  this.edad = edad;
  this.profesion = profesion;
}
// Agregar un método al prototipo de la función creada anteriormente
Persona.prototype.saludar = function() {
  console.log(`Hola, soy ${this.nombre} y tengo ${this.edad} años.`);
};
// Instanciar a la clase creada
let persona1 = new Persona("Juan", 30, "Desarrollador");
let persona2 = new Persona("María", 25, "Diseñadora");

// Llamar al método de la clase
persona1.saludar(); // Imprime Hola, soy Juan y tengo 30 años.
persona2.saludar(); // Imprime Hola, soy María y tengo 25 años.
```

Ejemplo 2:

```
function Persona(nombre, edad) {
  this.nombre = nombre;
  this.edad = edad;
}

Persona.prototype.saludar = function() {
  console.log("Hola, mi nombre es " + this.nombre + " y tengo " + this.edad + " años.");
};
```

```
var juan = new Persona("Juan", 25);
juan.saludar();
```

El resultado de la de instanciar al objeto y llamar al objeto es: “Hola, mi nombre es Juan y tengo 25 años.”

7.5 Propiedades con GET y SET en JavaScript

Los métodos de get y set permiten controlar tanto la definición de los valores y el acceso a estos cuando están dentro de un objeto, en el desarrollo estas funcionalidades son de vital importancia al permitir el acceso, validación y asignación de datos basándose en estas variables.

Ejemplo:

```
// Definir un objeto
class Persona {
  constructor(nombre, edad) {
    this._nombre = nombre; // La propiedad real es _nombre
    this._edad = edad;
  }
  // Definir un método getter para acceder a la propiedad nombre
  get nombre() {
    console.log("Obteniendo el nombre");
    return this.nombre;
  }
  // Definir un método setter para la acceder a la propiedad nombre
  set nombre(nuevoNombre) {
```

```
    if (typeof nuevoNombre === 'string') {
      console.log("Asignando nuevo nombre");
      this._nombre = nuevoNombre;
    }
  }
```

7.6 Ejercicios

Definir un objeto llamado mascota con varias propiedades propias del objeto incluyendo un objeto dentro con propiedades del dueño y mostrar un dato de la mascota y un dato del dueño.

```
let mascota = {
  nombre: "Scott",
  raza: pastor,
  dueño: {
    nombre_d: "Mario",
    ciudad: "Quito",
  }
}
console.log(mascota.nombre);
console.log(mascota.dueño.nombre_d);
```

En el siguiente ejemplo que se presenta se crea un objeto persona con atributos edad, dirección y cedula, imprimirlos cedula mediante la propiedad get

```
class persona {
  constructor(edad,dirección,cedula) {
```

```
    this.edad =edad;
    this.direccion =direccion;
this.cedula = cedula;
  }
  get cedula () {
    console.log("Esta es la cedula");
    return this.cedula;
  }
```

ISTPET
INSTITUTO TRAVERSARI

CAPÍTULO 8

Librerías de clases y objetos en JavaScript

CAPÍTULO 8

Librerías de clases y objetos en JavaScript

8.1. Objetos incorporados en JavaScript

Los objetos incorporados en JavaScript son elementos fundamentales que forman parte del entorno de ejecución del lenguaje. Estos objetos proporcionan funcionalidades esenciales y están disponibles globalmente en el entorno del navegador o en entornos de ejecución de JavaScript, como Node.js. Algunos ejemplos de objetos incorporados incluyen Object, Array, String, Number, Date, y otros.

Ejemplo 1:

```
// Uso del objeto Array incorporado en JavaScript

var miArray = [1, 2, 3, 4, 5];

console.log(miArray.length);

// Muestra la longitud del array: 5

console.log(miArray.join(", "));

// Convierte el array a una cadena separada por comas: 1, 2, 3, 4, 5
```

Ejemplo 2:

- **String:** Se puede trabajar con cadenas de caracteres. Por ejemplo, Usando el método toUpperCase() para convertir una cadena a mayúsculas: "hola".toUpperCase().

  El resultado serio “HOLA”

- **Math:** Se puede trabajar con funciones matemáticas. Por ejemplo, usar el método sqrt() para calcular la raíz cuadrada de un número: Math.sqrt(16).
- **Number:** Para realizar algunas cosas con números. Por ejemplo, puedes usar el método toFixed() para redondear un número a un número específico de decimales: (3.14159).toFixed(2).

8.2. Clase Date en JavaScript

La clase “Date” en JavaScript se utiliza para trabajar con fechas y horas. Proporciona métodos para la creación de objetos de fecha, manipulación y obtención de componentes específicos de la fecha y hora.

Ejemplo:

```
// Uso de la clase Date en JavaScript

var fechaActual = new Date();

var mesActual = fechaActual.getMonth() + 1;

// Suma 1 para obtener el mes en un rango de 1 a 12

console.log(mesActual);

// Muestra el mes actual en formato convencional (1-12)
```

8.3. Clase Math en JavaScript

La clase “Math” en JavaScript proporciona funciones matemáticas estáticas que se utilizan para re alizar operaciones comunes, como redondeo, potenciación, funciones trigonométricas, entre otras.

```
// Uso de la clase Math en JavaScript

var numero = 5.678;

console.log(Math.round(numero));

// Redondea al número entero más cercano: 6

console.log(Math.sqrt(numero));

// Obtiene la raíz cuadrada: 2.3848480035423646

console.log(Math.sin(Math.PI / 2)); // Calcula el seno de 90 grados: 1
```

8.4. Clase Number en JavaScript

La clase ‘’Number’’ en JavaScript es un objeto que proporciona funciones y propiedades para trabajar con valores numéricos. Se puede usar esta clase para realizar operaciones matemáticas y manipular números.

```
//Uso de la clase Number en JavaScript

//Creamos la instancia de nuestra clase

Let num1 = new Number(42);   //Utilizamos el operador new

Let num2 = 123; //Utilizamos un valor numérico primitivo

//Realizar operaciones con la clase Number

let suma = num1 + num2;

console.log(“Suma”, suma); //Resultado 165

//Utilizamos la propiedad y métodos de la clase Number

Console.log(“Valor máximo: ”, Number.MAX_VALUE); //Valor Máximo representable

Console.log(“Entero:”, num1.isInteger()); //Verificar si el número es entero
```

8.5. Clase Boolean en JavaScript

La clase ‘’Boolean’’ en JavaScript se utiliza para representar valores lógicos, y se puede utilizar para realizar operaciones lógicas y evaluar condiciones en el código.

```
//Uso del Boolean en JavaScript

//Creamos las instancias de la clase Boolean

Let bool1 = new Boolean(true); //Utilizamos un operador new

Let bool2 = false; //Utilizamos un valor booleano primitivo
```

```
//Realizamos operaciones lógicas con la clase Boolean

Let resultadoAND = bool1 && bool2;

Console.log("Resultado AND:", resultadoAND); //Resultado: false

//Utilizamos propiedades y métodos de la clase Boolean

Console.log("Valor primitivo:", bool1.valueOf()); //Obtenemos el valor primitivo
```

8.6. Ejercicios

Ejercicio: *Cristian va a caminar desde su casa hasta el veterinario, y quiere calcular cuántos metros recorrerá en total.*

Código:

```
//Insertar datos de Cristian

Let distanciaCasaVeterinario = 2.5; //Kilometros

//Conversion de kilómetros a metros

Let distanciaMetros = distanciaCasaVeterinario * 1000;

Console.log('Cristian va a recorrer ${distanciaMetros} metros hasta el veterinario. ');
```

Ejercicio:

Briana quiere programar las próximas clases de su curso online y necesita saber la fecha exacta de la próxima lección.

Código:

```
//Datos para programar la próxima clase
Let diasParaProximaClase = 7;
//Calcular la fecha de la próxima clase
Let fechaProximaClase = new Date();
fechaProximaClase.setDate(fechaProximaClase.getDate() + diasParaProximaClase);
//Obtener los componentes de la fecha
Let diaProximaClase = fechaProximaClase.getDate();
```

```
Let mesProximaClase = fechaProximaClase.getMonth();
Let anioProximaClase = fechaProximaClase.getFullYear();
Console.log(‘La próxima clase de Briana será el ${diaProximaClase} /${mesProximaClase} / ${anioProximaClase}.’);
```

ISTPET
INSTITUTO TRAVERSARI

CAPÍTULO 9

Eventos en JavaScript

CAPÍTULO 9

Eventos en JavaScript

9.1. Los eventos en JavaScript

Los eventos de JavaScript son sucesos que ocurren en el sistema que estamos programando, como, por ejemplo, el clic de un usuario en un botón o el envío de un formulario. Para reaccionar a estos sucesos, podemos usar funciones de JavaScript llamadas manejadores de eventos, que se ejecutan automáticamente cuando se produce el evento correspondiente. Los eventos se disparan dentro de la ventana del navegador y normalmente están relacionados con algún elemento específico de la página web.

Por ejemplo:

- El usuario selecciona, hace clic o pasa el ratón por encima de cierto elemento.
- El usuario presiona una tecla del teclado.
- El usuario redimensiona o cierra la ventana del navegador.
- Una página web terminó de cargarse.
- Un formulario fue enviado.
- Un vídeo se reproduce, se pausa o termina.
- Ocurrió un error.

9.2. Los tipos de eventos en JavaScript

Evento change (onchange)

Se desata este evento cuando cambia el estado de un elemento de formulario, en ocasiones no se produce hasta que el usuario retira el foco de la aplicación del elemento.

Ejemplo:

```
<select onchange="alert('El opción se ha bloqueado)">
<option value="1">Opción 1</option>
<option value="2">Opción 2</option>
</select>
```

- **click (onclick)**

Se produce cuando se da una pulsación o clic al botón del ratón sobre un elemento de la página, generalmente un botón o un enlace.

Ejemplo:

```
<button onclick="alert('¿Tienes permisos para ir la siguiente página?')">Siguiente página</button>
```

- **keydown (onkeydown)**

Este evento se produce en el instante que un usuario presiona una tecla, independientemente que la suelte o no. Se produce en el momento de la pulsación.

Ejemplo:

```
<input type="text" onkeydown="alert('Haz pulsado la tecla A')">
```

- **keypress (onkeypress)**

Ocurre un evento onkeypress cuando el usuario deja pulsada una tecla un tiempo determinado. Antes de este evento se produce un onkeydown en el momento que se pulsa la tecla.

Ejemplo:

```
<input type="text" onkeypress="alert('Has pulsado la tecla A y la estás manteniendo pulsada')">
```

- **load (onload)**

Este evento se desata cuando la página, las imágenes, han terminado de cargarse.

Ejemplo:

```
<img src="paisaje.png" onload="alert('La imagen se ha cargado')">
```

- **mousemove (onmousemove)**

Se produce cuando el ratón se mueve por la página.

Ejemplo:

```
<div onmousemove="alert('El ratón esta encima de la imagen')">
 <img src="paisaje.png">
</div>
```

- **submit (onsubmit)**

Ocurre cuando el visitante apreta sobre el botón de enviar el formulario. Se ejecuta antes del envío propiamente dicho.

Ejemplo:

```
// Código JavaScript que se ejecuta antes de enviar el formulario
<form action="index.php" method="post">
 <input type="text" name="nombre">
 <input type="submit" value="Enviar" onclick="alert('El formulario se va a enviar')">
</form>
```

El atributo action tiene como fin ejecutar la función del archivo index.php con su respectivo método en este caso POST para envío de datos

9.3. Ejemplos de eventos en JavaScript Onabort

Este evento se produce cuando un usuario detiene la carga de una imagen, ya sea porque detiene la carga de la página o porque realiza una acción que la detiene, como por ejemplo irse de la página.

Ejemplos:

- Mostrar una advertencia al usuario si la imagen no se carga:

```
<img src="paisaje.png" onabort="alert('La imagen no ha podido cargar')">
```

- Reemplazar la imagen con una imagen alternativa:

```
<img src=" paisaje.png " onabort="this.src='paisaje2.jpg'">
```

9.4. Ejemplo de evento onblur en JavaScript

Se desata un evento onblur cuando un elemento pierde el foco de la aplicación. El foco de la aplicación es el lugar donde está situado el cursor, por ejemplo, puede estar situado sobre un campo de texto, una página, un botón o cualquier otro elemento.

Ejemplos:

```
<input type="text" id="correo" onblur="alert('Has salido del campo de correo)">
```

Este código vacía el campo de texto contraseña cuando el usuario sale de él.

```
<input type="text" id=" contraseña " onblur="this.value=" ">
```

9.5 Evento onunload de JavaScript

El evento JavaScript "onunload" se utiliza cuando el usuario navega para cambiar páginas en el mismo sitio o para recargar la página en algunas situaciones menos utilizadas. Se ejecuta el evento cuando estas acciones se identifican y realizan su

función asignada antes de que el usuario salga de la página. Comúnmente se utilizan para generar mensajes de salida, almacenar datos o animaciones. Pero también tiene restricciones de seguridad en eventos dependiendo de la funcionalidad que se tengo y el navegador, el evento "onunload" tiene acciones muy limitadas.

Ejemplo:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Retraso antes de Cambiar de Página</title>
    <style>
        body {
            transition: background-color 1s ease;
        }
    </style>
    <script>
        // Función que se ejecutará antes de descargar la página
        window.addEventListener('beforeunload', function (event) {
            document.body.style.backgroundColor = 'red';
    // Asegurar que el mensaje se muestra en navegadores basados en WebKit
            event.returnValue = '¡Estás a punto de salir de la página!';
        });

        // Función para cargar una nueva página con retraso
```

```
        function cargarNuevaPaginaConRetraso() {
            // Cambiar a rojo antes de cambiar de página
            document.body.style.backgroundColor = 'red';

            // Esperar 5 segundos antes de redireccionar
            setTimeout(function () {
                window.location.href = 'index.html'; //Otra pagina dentro del sitio web
            }, 5000);
        }
    </script>
</head>
<body>
    <p>Contenido de la página.</p>
    <!-- Botón para cargar una nueva página con retraso -->
    <button onclick="cargarNuevaPaginaConRetraso()">Cargar Nueva Página con Retraso</button>
</body>
</html>
```

9.6. Evento onload de JavaScript

Este evento de JavaScript “onload” se lo utiliza para ejecutar funciones después de que la pagina se ha cargado completamente con sus recursos. Es muy útil para realizar mensajes de bienvenida, animaciones. Con este evento nos facilita tener acciones únicas que podemos implementar dentro de la página con funciones personalizadas.

Ejemplo:

```
<!DOCTYPE html>
<html lang="es">
```

```
<head>
    <meta charset="UTF-8">
    <title>Evento onload</title>
    <script>
        // Función que se ejecutará cuando la página se haya cargado completamente
        window.onload = function() {
            mostrarMensajeDeBienvenida();
        };

        // Función para mostrar un mensaje de bienvenida
        function mostrarMensajeDeBienvenida() {
            alert("¡Bienvenido! La página se ha cargado completamente.");
        }
    </script>
</head>
<body>
    <p>Contenido de la página.</p>
</body>
</html>
```

9.7. Eventos personalizados en JavaScript

Para los eventos personalizados en JavaScript tiene su palabra reservada y una sintaxis que seguir, y para crearlo se usa el “CustomEvent” el cual crear y disparar eventos personalizados. La utilizada de estos eventos es para comunicar o notificar sobre acciones personalizadas en la página mejorando la interactividad, con estos eventos Podemos definir su comportamiento y la información que vamos a mostrar, con el fin de que tenga una adaptabilidad perfecta con los requerimientos de los usuarios/clientes.

Ejemplo:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>Evento Personalizado</title>
</head>
<body>
   <!-- Agregamos un elemento con el id "miElemento" -->
   <button id="miElemento">Haz clic aquí para disparar el evento
personalizado</button>

   <script>
      // Creamos un nuevo evento personalizado llamado "miEvento"
      var miEvento = new CustomEvent("miEvento", {
```

```
            detail: {
                mensaje: "¡Hola desde mi evento personalizado!",
            },
        });
        // Creamos una función para manejar el evento personalizado
        function manejarEvento(event) {
            alert(event.detail.mensaje);
            console.log(event.detail.mensaje);
        }

        // Añadimos el evento personalizado al elemento deseado
        var miElemento = document.getElementById("miElemento");
        miElemento.addEventListener("miEvento", manejarEvento);

        // Disparamos el evento personalizado al hacer clic en el elemento
        miElemento.addEventListener("click", function() {
            miElemento.dispatchEvent(miEvento);
        });
    </script>
</body>
</html>
```

9.8. Ejercicios

Ejercicio: *Crea un programa que muestre un mensaje cuando el usuario seleccione una opción de un menú desplegable.*

Código:

```
<!DOCTYPE html>
<html lang="es">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
</head>
<body>
   <select id="miSelect" onchange="mostrarMensaje()">
      <option value="1">Opción 1</option>
      <option value="2">Opción 2</option>
    </select>
    <p id="mensaje"></p>

    <script>
      function mostrarMensaje() {
        var x = document.getElementById("miSelect").value;
        document.getElementById("mensaje").innerHTML = "Seleccionaste la opción "
+ x;
```

```
    }
  </script>
</body>
</html
```

Ejercicio: *Desarrolla un programa que muestre un mensaje de bienvenida personalizado según el nombre que el usuario escriba en un campo de texto, y que al cargar la página reproduzca un sonido.*

Código:

```
<!DOCTYPE html>
<html lang="es">
<head>
</head>
<body>
  <label for="">Nombre</label>
  <input type="text" id="nombre" onchange="mostrarSaludo()">
  <p id="saludo"></p>

  <script>
    function mostrarSaludo() {
      var nombre = document.getElementById("nombre").value;
      document.getElementById("saludo").innerHTML = "¡Hola, " + nombre + "!";
    }
```

```
        //Funcion para reproducir el audio
    window.onload = function() {
      var audio = new Audio("ruta/al/sonido.mp3");
      audio.play();
    };
  </script>

</body>
</html>
```

ISTPET
INSTITUTO TRAVERSARI

CAPÍTULO 10

NODE.JS

CAPÍTULO 10

NODE.JS

10.1 Antecedentes

Node.js fue creado por Ryan Dahl en 2009, marcando el comienzo de una nueva era en el desarrollo de JavaScript en el lado del servidor. La introducción de Node Package Manager (NPM) en 2010 fue un hito significativo, ya que facilitó la gestión de paquetes de Node.js. A lo largo de los años, Node.js ha visto varias mejoras de rendimiento, con lanzamientos notables como Node.js v0.12 en 2014, y el establecimiento de la Fundación Node.js en 2015 para supervisar su desarrollo. El lanzamiento de Node.js v10 en 2018 trajo más mejoras en rendimiento y estabilidad. Se adoptó una nueva cadencia de lanzamiento en 2019, con lanzamientos de soporte a largo plazo (LTS) cada seis meses y lanzamientos de nuevas funciones cada cuatro meses. El lanzamiento de Node.js 16 LTS en 2020 incluyó muchas nuevas funciones y mejoras, y lanzamientos posteriores como Node.js 17 continuaron mejorando el rendimiento y la estabilidad. Node.js se ha utilizado para construir aplicaciones para importantes empresas como PayPal, Netflix, eBay, Walmart, Uber, NASA, Microsoft, Yahoo!, Amazon y General Electric.

10.2. Comando Node

El comando Node se utiliza para ejecutar código JavaScript fuera de un entorno de navegador. Permite a los desarrolladores ejecutar scripts en el lado del servidor, lo que permite el desarrollo de aplicaciones de red escalables. Node.js funciona con una arquitectura no bloqueante y basada en eventos, lo que lo hace eficiente para manejar operaciones concurrentes sin crear múltiples hilos.

Por ejemplo, si el archivo que contiene el código principal de la aplicación se llama app.js, se puede ejecutar escribiendo node app.js en la línea de comandos. También se puede usar la opción -e o --eval "script" para evaluar una expresión JavaScript directamente desde la línea de comandos, sin necesidad de crear un archivo aparte.

10.3. Módulos

Los módulos en JavaScript son piezas de código reutilizables que se pueden exportar desde un archivo e importar en otro. Este enfoque modular ayuda en la organización y mantenimiento del código. En Node.js, los módulos se pueden importar utilizando la declaración de importación con una ruta al archivo del módulo, y se pueden exportar utilizando la declaración de exportación. Por ejemplo, una función show_message se puede exportar desde un módulo y luego importar y usar en otro archivo.

10.4. Módulo de sistema de archivos

El módulo File System (fs) en Node.js proporciona funcionalidad para interactuar con el sistema de archivos en su computadora. Incluye métodos para leer y escribir archivos, crear directorios y otras tareas de administración de archivos. Sin embargo, al usar herramientas como webpack, los desarrolladores pueden encontrar problemas como "No se puede encontrar el módulo 'fs'". Para resolver esto, se pueden agregar configuraciones a webpack para evitar la agrupación de fs para el navegador.

10.5. Módulo HTTP

El módulo HTTP de Node.js es una herramienta esencial para crear servidores web y API, ya que permite procesar solicitudes HTTP y enviar respuestas. Este módulo, parte del núcleo de Node.js y escrito en C para optimizar el rendimiento de las conexiones web, ofrece una variedad de métodos y propiedades para controlar diferentes aspectos de los protocolos HTTP. Para utilizarlo, se debe importar con el método require(). A través del método createServer() se puede crear un servidor HTTP que escuche en un puerto específico y responda a las solicitudes entrantes. Además, el servidor HTTP creado con este módulo puede atender diversas solicitudes HTTP, como GET, POST, PUT, DELETE, entre otras. También provee métodos y propiedades para controlar aspectos como el manejo de cabeceras HTTP, cookies, sesiones y archivos estáticos.

10.6. File server

Un servidor de archivos es un programa que permite a los usuarios acceder a archivos almacenados en un sistema informático. Node.js proporciona una API para crear servidores de archivos.

Para crear un servidor de archivos simple, podemos usar el módulo fs para leer y escribir archivos. El siguiente código crea un servidor de archivos que sirve todos los archivos en el directorio actual:

Ejemplo 1:

```
const fs = require('fs');


const server = http.createServer((req, res) => {
  const filePath = req.url;
  const fileContent = fs.readFileSync(filePath);

  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end(fileContent);
});

server.listen(8080);
```

Este código crea un servidor HTTP que escucha en el puerto 8080. Cuando un cliente envía una solicitud a /, el servidor leerá el archivo con el mismo nombre que la solicitud y lo devolverá al cliente.

10.7 Summary

Node.js es un entorno de ejecución JavaScript multiplataforma que permite a los desarrolladores crear aplicaciones web, servidores de archivos y otras aplicaciones.

Se basa en una arquitectura no bloqueante y basada en eventos, lo que lo hace eficiente para manejar operaciones concurrentes.

Node.js se puede usar para ejecutar scripts, evaluar expresiones, crear aplicaciones, importar y exportar módulos, interactuar con el sistema de archivos e incluso crear servidores web. Es una herramienta poderosa y versátil que puede ser utilizada por desarrolladores de todos los niveles de experiencia.

10.8 Ejercicios

Ejercicio: *Crea un servidor de archivos que sirva todos los archivos en un directorio específico.*

Código:

```
const fs = require('fs');

const server = http.createServer((req, res) => {
  const filePath = req.url;
  const fileContent = fs.readFileSync(filePath);
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end(fileContent);
});

server.listen(8080);
```

Ejercicio: *Crea un servidor de archivos que sirva archivos con un formato específico.*

Código:

```
const fs = require('fs');


const server = http.createServer((req, res) => {
```

```
  const filePath = req.url;
  const fileExtension = filePath.split('.')[1];

  if (fileExtension === 'txt') {
    const fileContent = fs.readFileSync(filePath);

    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end(fileContent);
  } else {
    res.writeHead(404);
    res.end('El archivo no existe');
  }
});

server.listen(8080);
```

Ejercicio: *Crea un servidor de archivos que permita a los usuarios subir y descargar archivos.*

Solución

```
const fs = require('fs');
const multer = require('multer');
```

```
const server = http.createServer((req, res) => {
  if (req.method === 'POST') {
    const upload = multer({ dest: './uploads' });
    const file = upload.single('file');

    file.on('progress', (progress) => {
      console.log(El archivo se está subiendo... ${progress});
    });

    file.on('error', (err) => {
      console.log(Ocurrió un error al subir el archivo: ${err});
      res.writeHead(500);
      res.end('Ocurrió un error al subir el archivo');
    });

    file.on('end', () => {
      console.log(El archivo se subió correctamente);
      res.writeHead(200);
      res.end('El archivo se subió correctamente');
    });

    file.upload();
```

```
  } else if (req.method === 'GET') {
    const filePath = req.url;
    const fileContent = fs.readFileSync(filePath);

    res.writeHead(200, { 'Content-Type': 'application/octet-stream' });
    res.end(fileContent);
  } else {
    res.writeHead(404);
    res.end('El método no es válido');
  }
});

server.listen(8080);
```

ACERCA DE LOS AUTORES

CRISTIAN ANDRÉS COLA PÉREZ

Ingeniero en Informática y Sistemas Computacionales graduado de la Universidad Técnica de Cotopaxi y con un título de Máster Universitario en Dirección e Ingeniería de Sitios Web obtenido en la UNIR, desempeñando roles destacados a lo largo de su carrera. Su experiencia incluye importantes responsabilidades como docente y creador de planos de estudio innovadores, integrando las últimas tendencias y tecnologías en desarrollo de software. Ha ejercido estas funciones tanto en el Tecnológico Universitario Vida Nueva como en el Instituto Superior Tecnológico Mayor Pedro Traversari.

Su compromiso con la vinculación entre la educación superior y la comunidad se evidencia en la creación de sólidos lazos institucionales. Además, ha ocupado el cargo de Coordinador de Carrera de Desarrollo de Software en ambas instituciones, donde ha trabajado activamente para promover oportunidades de desarrollo profesional y supervisar el progreso académico de los estudiantes.

Como investigador entusiasta, se dedica a la exploración continua de nuevas tecnologías, métodos y soluciones, con el objetivo de contribuir al avance constante en el campo tecnológico.

JOSUÉ EMANUEL TELLO MONTERO

Ingeniero de Desarrollo de Software, con trayectoria en docencia de 2 años, experiencia en el campo profesional del desarrollo de software de 1 año, autor de artículos científicos publicados, conferencista y organizador de eventos académicos universitarios; participante y ganador de concursos locales y nacionales en desarrollo de software, organizados por la Escuela Superior Politécnica de Chimborazo; catedrático y colaborador de actividades académicas, impartiendo clases tipo espejo a favor de los estudiantes de la ESPOCH.

FRANCISCO XAVIER MESÍAS FLORES

Ingeniero de Software con título obtenido en la Escuela Superior Politécnica de Chimborazo (ESPOCH). Cuenta con experiencia en diversas áreas tecnológicas, incluyendo el desarrollo de encuestas provinciales para análisis de consumo en Chimborazo. Especializado en el desarrollo y aplicación de aplicaciones web, desde sistemas CMS hasta frameworks avanzados. Experto en ciberseguridad, con destacada participación en pruebas de seguridad para aplicaciones mHealth, implementando medidas efectivas para proteger datos sensibles.

Centro de Investigación y Desarrollo Profesional
Generador de Conocimientos...

ISTPET
INSTITUTO TRAVERSARI

ISBN: 978-9942-607-75-1

CIDE Editorial