

# DESARROLLO BACK END -INTERMEDIO

Autor de contenido

**Luz Liliana Herrera Polo**



# Tabla de Contenido



## Presentación

En el presente documento está presentada de manera detallada la propuesta a desarrollar con el curso Desarrollador Backend, ofrecidos principalmente a personas con educación media culminada que vivan en Colombia y que estén interesados en aprender en el campo de las tecnologías de la información. Con este curso se busca que el estudiante al finalizar la formación sea capaz de desarrollar aplicaciones backend robustas implementando lógica de programación con JavaScript, bases de datos y sistemas de control de versiones.

## Objetivos del curso (competencias)



### Objetivo general

Aprender a desarrollar aplicaciones modernas para procesar, almacenar y proteger la información de un programa de manera robusta y escalable. Implementando buenas prácticas y la lógica de programación con JavaScript.

### Objetivo específico

- Por medio del pensamiento lógico, el lenguaje de programación y las bases de datos diseñar y desarrollar aplicaciones de software que se ajusten a los requerimientos planteados en cualquier problema.
- Organizar la realización de pruebas que verifiquen el correcto funcionamiento de las aplicaciones de software desarrolladas y verificando también que el desarrollo se ajusta a los requisitos de análisis y diseño.
- Utilizar un sistema de control de versiones con el fin de gestionar los cambios en el código fuente a lo largo del tiempo.

## Objetivos del curso (competencias)



Dar el conocimiento necesario para entender cómo funciona el backend y la programación por medio de JavaScript

## Mapa de contenido de la unidad

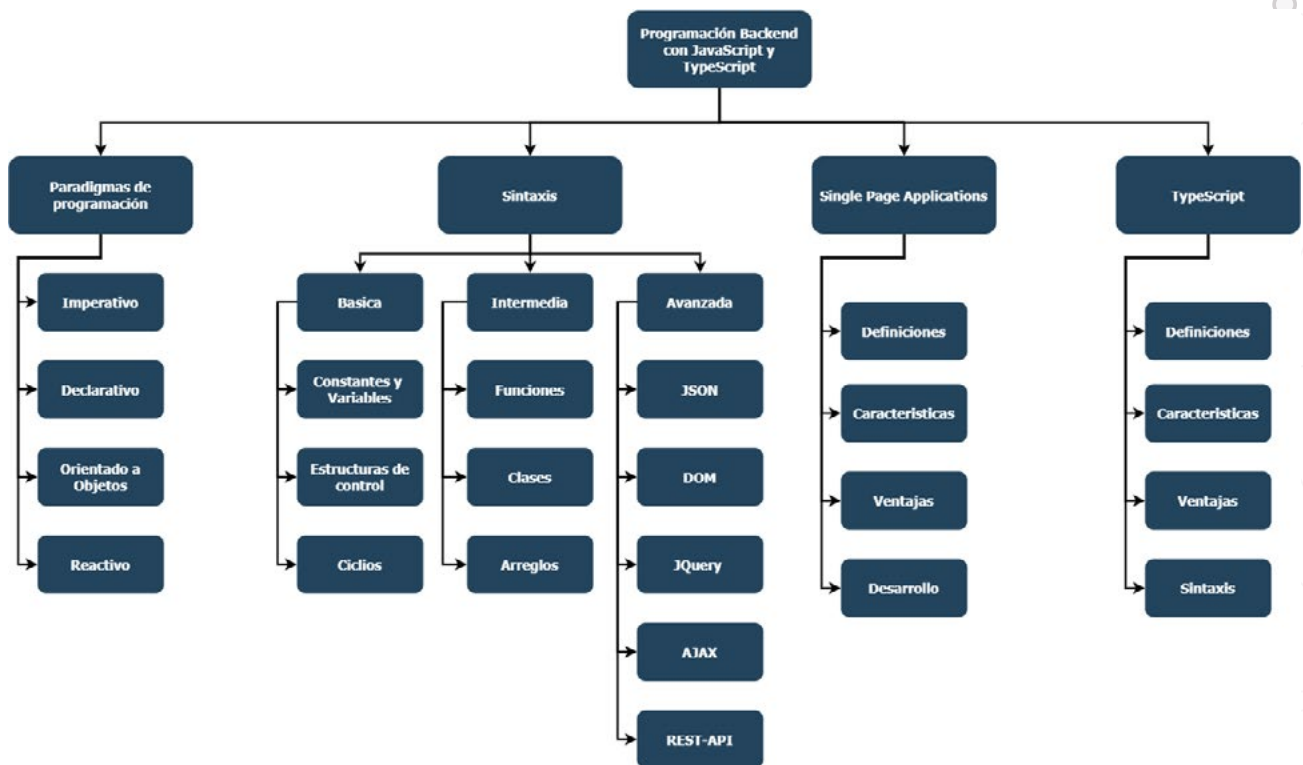


Figura 1. Mapa de contenido del módulo 6

## Módulo 6

# Programación Backend con JavaScript

### 6.1. Paradigmas de programación

Clase teórica en donde se explican los distintos paradigmas de programación (imperativo, declarativo, orientado a objetos y reactivo) y cuales son los que se van a realizar en la práctica en el entorno de desarrollo del curso.

#### ¿Qué es un paradigma de programación?

Un paradigma de programación es un estilo o diseño de cómo se programa el software. Se define como un conjunto de métodos sistemáticos, estándares y objetivos de rendimiento que se deben cumplir para lograr codificar en el lenguaje para el cual se realizó y estructuró el paradigma, a continuación veremos diferentes tipos de paradigmas que nos ayudarán a entender su necesidad.

- Programación Imperativa:** La programación imperativa es el paradigma clásico secuencial de los primeros lenguajes de programación, su objetivo es que el código se codifique en instrucciones ordenadas que deben tener sentido secuencial, gracias a esto se obtiene un código fácil de leer y mantener.

- Programación Declarativa:** La programación declarativa se basa en siempre esperar el “qué” y no el “cómo”, basándose en la descripción del resultado, esto genera menos líneas de código pero una gran abstracción del código lo que provoca que sea más difícil de leer y entender.

- Programación Orientada a Objetos:** La programación orientada a objetos construye un modelo de objetos que representan los elementos del problema a resolver. Este paradigma se caracteriza por separar los diferentes elementos y componentes de un programa para así disminuir los errores y proporcionar reutilización de código.

- Programación Reactiva:** La programación reactiva, se enfoca principalmente en proyectos de software con flujos de datos finitos o infinitos de manera asíncronica. Los sistemas reactivos deben ser responsivos, resilientes, elásticos y orientados a mensajes.

## 6.2. Sintaxis básica

Clases teórico prácticas que se centran en dar los primeros pasos en la programación, en donde se aplicarán en el entorno de desarrollo conceptos como variables, constantes, tipos de datos, operadores (aritméticos, de comparación, lógicos, de concatenación).

### Comentarios

Los comentarios en el entorno del desarrollo son comúnmente usados para hacer anotaciones o apuntes sobre el código, solo son visibles para los desarrolladores.

```
1  //Esto es un comentario de una linea
2
3  /*
4  Esto es un bloque
5  de comentarios, es decir,
6  de varias lineas
7  */
```

Figura 6. Comentarios. Fuente: Elaboración propia.

### Variables y constantes

En el desarrollo de software las variables pueden ser declaradas en un ámbito global o local. La declaración de variables en el ámbito global, se refiere a que la variable puede ser declarada fuera de un bloque de código (trozo de código delimitado por {}) y se accede a esta variable desde cualquier bloque del código, mientras que la declaración de variables en el ámbito local hace referencia a las variables que solo están disponibles para su uso dentro de ese bloque.

Para declarar variables y constantes en JavaScript se utilizan las siguientes palabras reservadas:

- var:** Permite declarar una variable en el ámbito global.
- let:** Permite declarar una variable en el ámbito local.
- const:** Permite declarar una constante (no cambia en toda la ejecución del código).

Para entender mejor tenemos el siguiente ejemplo. Observe el siguiente código.

```
1 var variableGlobal = "Esto es una variable global"
2
3 {
4   let variableLocal = "Esto es una variable local"
5   console.log(`Llamado variable local dentro del bloque ${variableLocal}`)
6 }
7
8 console.log(variableGlobal)
9 console.log(`Llamado variable local fuera del bloque ${variableLocal}`)
10
```

Lo que arroja en consola lo siguiente.

CONSOLE X

Llamado variable local dentro del bloque Esto es una variable local  
Esto es una variable global

► ReferenceError: variableLocal is not defined  
at <anonymous>:8:56  
at dn (<anonymous>:16:5449)

El llamado de la variable local (variableLocal) dentro del bloque de código fue exitoso y arrojó el mensaje esperado, el llamado de la variable global (variableGlobal) también arroja el mensaje esperado, pero en cambio, el llamado de la variable local (variableLocal) fuera del bloque arrojó un error el cual quiere decir que la variable local no fue definida, esto sucede porque la variable local no existe fuera de bloque en el que fue declarada.

Las constantes se declaran de la misma manera, por favor observe el siguiente código como ejemplo.

```
1 const PI = 3.1416
2 console.log(`El valor de PI es ${PI}`)
```

Lo cual arroja en consola el siguiente resultado.

CONSOLE X

El valor de PI es 3.1416

## Tipos de datos

•**Boolean:** Representa un valor true o false. Se define de la siguiente manera.

```
2
3   let datoBooleano: boolean
4   let datoBooleano = true; //Definición
```

•**Undefined:** Representa variables que solo han sido declaradas y no se les ha asignado un valor.

•**Number:** Número entero o decimal.

```
1
2   let datoNumerico: number
3   let datoNumerico=15.8; //Definición
```

•**String:** Representa una secuencia de caracteres.

```
1
2   let datoString: string
3   let datoString="Hola mundo!"; //Definición
```

## Operadores

•**Asignación (=):** Es el operador más utilizado, se utiliza para asignar valores a las variables y constantes.

```
1   let variable="Asignación valor a la variable"
```

•**Decremento e incremento (-- / ++):** Operador utilizado para incrementar o decrementar el valor de una variable numérica en una unidad. Estos operadores sólo son válidos para variables numéricas. Como se ve a continuación en el siguiente código.

```
1   var num=10
2   num++ //Incremento
3   console.log(`Variable incrementada ${num}`)
4
5   var num1=15
6   num1-- //Decremento
7   console.log(`Variable decrementada ${num1}`)
```



Lo que devuelve en consola lo siguiente.

CONSOLE X

Variable incrementada 11

Variable decrementada 14

•**Lógicos:** Son los operadores que se utilizan para la toma de decisiones en los algoritmos de acuerdo a ciertas condiciones que se le den. El valor que devuelven estos operadores siempre es booleano.

**-Negación (!):** Operador lógico utilizado para obtener el valor contrario de una variable booleana.

```
1 let variable = true
2 let variableNegada=!valor
3 console.log(`La variable negada es: ${variableNegada}`)
```

CONSOLE X

La variable negada es: true

**-AND (&&):** Operador lógico que devuelve True únicamente si todas las condiciones se cumplen. Como se ve en el siguiente ejemplo si tenemos un valor verdadero y otro falso, el resultado es falso.

```
1 let valor1 = true
2 let valor2 = false
3 let resultado= valor1 && valor2
4 console.log(`El resultado es: ${resultado}`)
```

CONSOLE X

El resultado es: false

Pero si cambiamos los dos valores a verdadero el resultado será verdadero

```
1 let valor1 = true
2 let valor2 = true
3 let resultado= valor1 && valor2
4 console.log(`El resultado es: ${resultado}`)
```

CONSOLE X

El resultado es: true

**-OR (||):** Operador lógico que devuelve True si alguna de las condiciones dadas se cumple.

```
1 let valor1 = true
2 let valor2 = false
3 let resultado= valor1 || valor2
4 console.log(`El resultado es: ${resultado}`)
```

CONSOLE X

El resultado es: true

**•Aritméticos:** Permiten realizar operaciones matemáticas. Los operadores que se utilizan más comúnmente son: suma (+), resta (-), multiplicación (\*) y división (/).

```
1 let num1 = 14
2 let num2 = 16
3
4 let suma = num1 + num2
5 let resta = num1 - num2
6 let multiplicacion = num1 * num2
7 let division = num1 / num2
8
9 console.log(`Resultado suma: ${suma}`)
10 console.log(`Resultado resta: ${resta}`)
11 console.log(`Resultado multiplicación: ${multiplicacion}`)
12 console.log(`Resultado división: ${division}`)
```

CONSOLE X

Resultado suma: 30  
Resultado resta: -2  
Resultado multiplicación: 224  
Resultado división: 0.875

•**Comparación:** Son los operadores que comparan el valor de variables numéricas para devolver un valor booleano. Estos operadores son los mismos que se ven comúnmente en matemáticas mayor que (>), menor que (<), mayor o igual (>=), menor o igual (<=), igual que (==) y distinto de (!=).

```
1  let num1 = 14
2  let num2 = 16
3
4  let mayor = num1 > num2
5  let menor = num1 < num2
6  let mayorQue = num1 >= num2
7  let menorQue = num1 <= num2
8  let igual = num1 == num2
9  let diferente = num1 != num2
10
11 console.log(`Mayor: ${mayor}
12 Menor: ${menor}
13 Mayor que: ${mayorQue}
14 Menor que: ${menorQue}
15 Igual: ${igual}
16 Diferente: ${diferente}`)
```

#### CONSOLE X

```
Mayor: false
Menor: true
Mayor que: false
Menor que: true
Igual: false
Diferente: true
```

## 6.3. Estructuras de control

Explicación y práctica de las estructuras de control utilizadas para controlar el flujo del programa, en las clases se desarrollarán los dos tipos de estructuras de control. El primer tipo que permite la ejecución condicional de bloques de código y que son conocidas como estructuras condicionales y por otro lado, las estructuras iterativas que permiten la repetición de un bloque de instrucciones, un número determinado de veces o mientras se cumpla una condición.

### Estructuras de control

Las estructuras de control en el código ayudan a controlar el flujo de operaciones durante la ejecución del programa. En JavaScript encontramos las siguientes estructuras de control.

•**Estructuras de control condicionales:** Son estructuras de control que permiten que se realice una instrucción u otra de acuerdo a la evaluación de una condición. Para estas instrucciones se utilizan las palabras reservadas `if` / `else`.

```
1 let num1 = 14
2 let num2 = 16
3
4 if(num1 > num2){
5   console.log(`El ${num1} es mayor que ${num2}`)
6 } else {
7   console.log(`El ${num1} es menor que ${num2}`)
8 }
```

CONSOLE X

El 14 es menor que 16

•**Estructuras de control condicionales anidadas:** Son aquellas estructuras que se necesita que revisen más de una condición.

```
1 let nota = 6
2
3 if(nota > 8){
4   console.log("Sobresaliente!")
5 } else if (nota > 6 && nota < 8){
6   console.log("Muy bien!")
7 } else {
8   console.log("Debes estudiar un poco más")
9 }
```

CONSOLE X

Debes estudiar un poco más

También se puede ejecutar las palabras reservadas switch / case como se ve a continuación.

```
1  let fruta="Manzana"
2
3  switch (fruta) {
4    case "Naranja":
5      console.log('El kilogramo cuesta 3.000')
6      break;
7    case "Fresa":
8      console.log('El kilogramo cuesta 4.000')
9      break;
10   case "Manzana":
11     console.log('El kilogramo cuesta 5.000')
12     break;
13   default:
14     console.log("No tenemos esa fruta")
15     break;
16 }
```

CONSOLE X

El kilogramo cuesta 5.000

•**Estructuras de control iterativas:** Son una secuencia de instrucciones orientadas a ejecutarse varias veces, son comúnmente llamadas ciclos.

-**For:** Permite ejecutar un código un número determinado de veces. Su sintaxis es la que encontramos a continuación.

```
1  var cont
2
3  for(cont=0; cont <=5; cont++){
4    console.log(`El valor del contador es: ${cont}`)
5  }
```

CONSOLE X

El valor del contador es: 0

El valor del contador es: 1

El valor del contador es: 2

El valor del contador es: 3

El valor del contador es: 4

El valor del contador es: 5

**-While:** Permite ejecutar un código mientras se cumpla una condición. Su sintaxis es la que vemos a continuación.

```
1  var num=0
2  //Imprime los números del 1 al 10
3  while(num<=10){
4      console.log(num)
5      num++
6  }
```

CONSOLE x

0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10

## 6.4. Funciones

Desarrollo y aplicación del concepto de funciones en el entorno de programación, se desea con esto que el estudiante aprenderá a manipular, organizar y reutilizar código de una manera acertada.

### ¿Qué es una función?

Las funciones son un conjunto de instrucciones de un programa que se centra en calcular o resolver un proceso de forma independiente del resto del programa, esto ayuda en el código y en el algoritmo a reutilizar procesos. Sus tres componentes principales son:

- Parámetros (valores de entrada de la función)
- Proceso (código de la función)
- Resultado (valor de retorno o valores de salida)

## Funciones declaradas

Las funciones declaradas, son funciones que forman parte del ámbito global del programa, que para que se ejecuten se debe hacer un llamado a la función, independientemente de donde se declaren se pueden ser llamadas desde cualquier parte del código. En JavaScript se definen mediante la palabra reservada `function`, seguida del nombre de la función, seguido de paréntesis en donde si se necesitan valores de entrada en la función (parámetros) se pueden poner dentro de los paréntesis seguidos de unas llaves que es donde va a ir el código de la función. Con la palabra reservada `return` definimos el valor que espera retornar la función cuando se ejecute. Para hacer el llamado a la función sólo es necesario escribir el nombre que se le dio a la función seguido de paréntesis con los argumentos de la función si la función los tiene. A continuación un ejemplo de una función con parámetros.

```
1  let num1=15
2  let num2=46;
3
4  function sumar (a,b){ //Definición de la función
5    | return console.log(a+b) //Valor de retorno
6  }
7
8  sumar(num1, num2) //Ejecución de la función
```

CONSOLE x

61

A continuación, un ejemplo de una función declarada sin parámetros.

```
1  function saludo(){ //Definición de la función
2    | return console.log("Hola mundo!") //Valor de retorno
3  }
4
5  saludo() //Ejecución de la función
```

CONSOLE x

Hola mundo!

## Funciones expresadas

La diferencia entre las funciones expresadas y declaradas es que mientras que las funciones declaradas hacen parte del ámbito local, las funciones expresadas hacen parte del ámbito concreto del programa, es decir, estas funciones no existen hasta que la expresión que las genera es ejecutada. La estructura de las funciones expresadas es similar a la de las funciones declaradas, pero, estas no inician con la palabra reservada function, tampoco es necesario que tengan un nombre y se pueden almacenar en una variable o constante. Para entender mejor ver los siguientes ejemplos.

```
1  const saludo =function(){ //Definición de la función
2  |  return console.log("Hola mundo!") //Valor de retorno
3  }
4
5  saludo() //Ejecución de la función
```

CONSOLE X

Hola mundo!

Pero, si intentamos llamar a la función saludo() antes de que la expresión que la genera es ejecutada nos aparece lo siguiente en consola.

```
1  saludo() //Ejecución de la función
2
3  const saludo =function(){ //Definición de la función
4  |  return console.log("Hola mundo!") //Valor de retorno
5  }
```

CONSOLE X

► TypeError: saludo is not a function  
at <anonymous>:8:1  
at dn (<anonymous>:16:5449)



Haciendo lo mismo con la función declarada, tenemos como resultado lo siguiente debido a que el ámbito de las funciones declaradas es global.

```
1  saludo() //Ejecución de la función
2
3  function saludo(){ //Definición de la función
4  |   return console.log("Hola mundo!") //Valor de retorno
5  }
```

CONSOLE X

Hola mundo!

## 6.5. Objetos

Con este concepto de objeto se da una introducción al estudiante a la programación orientada a objetos, se desea con esto que el estudiante sea capaz de abstraer el concepto de objeto usado en programación permitiendo separar los diferentes componentes de un programa, simplificando así su elaboración, depuración y posteriores mejoras.

### ¿Qué es un objeto?

Los objetos en JavaScript no son más que un conjunto de propiedades, en donde las propiedades tienen asociadas un nombre y un valor. Para entender mejor la definición de objeto ver el siguiente video <https://youtu.be/ykpT5P7171M>.

### Objetos literales

En JavaScript, los literales de los objetos son las llaves {}, dentro de las llaves se pueden declarar las propiedades del objeto separadas por coma y estas propiedades tienen un nombre y un valor. Para acceder al valor del atributo del objeto sólo es necesario usar la siguiente notación: nombreObjeto.nombreAtributo. Como en el siguiente ejemplo.

```
1  const persona = { // Definición del objeto persona
2    //Atributos
3    nombre: "Marco",
4    apellido: "Sandoval",
5    edad: 26,
6    hablar: function(){
7      return console.log(`Hola! Mi nombre es ${this.nombre}`)
8    }
9  }
10 //Acceder a la función hablar dentro del objeto persona
11 persona.hablar()
12 //Acceder a la propiedad edad del objeto persona
13 console.log(`Mi edad es ${persona.edad}`)
```

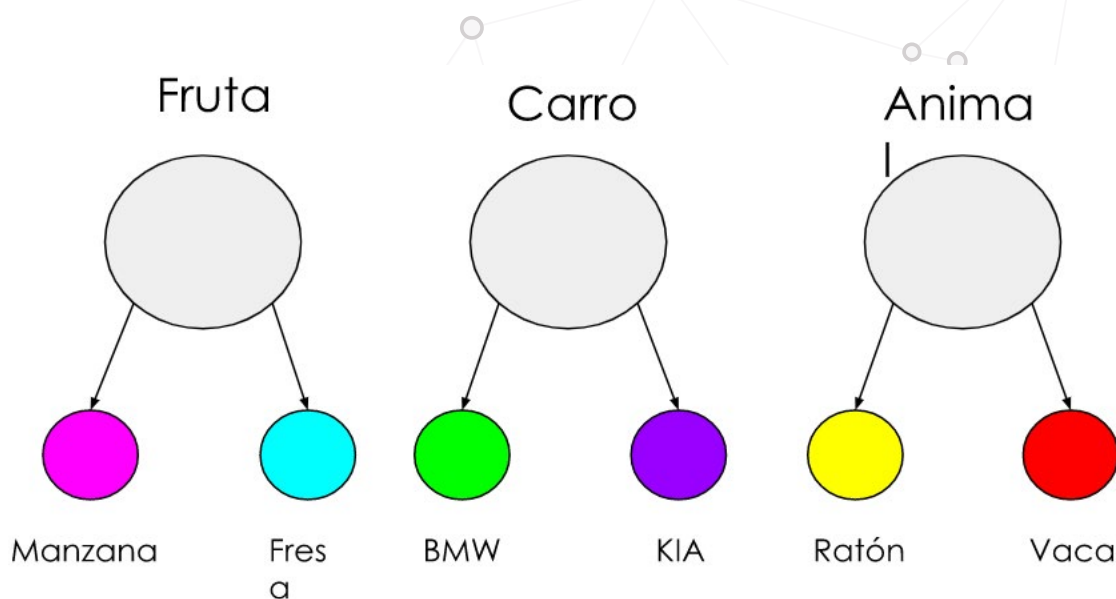
CONSOLE X

Hola! Mi nombre es Marco

Mi edad es 26

## Clases

Las clases en el desarrollo web no son más que una forma de organizar y simplificar el funcionamiento del código de una aplicación. Para entender mejor el concepto de clase y como se relaciona con los objetos por favor ver el siguiente ejemplo.



En la imagen anterior tenemos en primer lugar las clases (círculos grises). La clase es entonces un concepto abstracto de un objeto, y los objetos son los elementos que se basan en la clase. En el primer ejemplo se tienen dos objetos: manzana y fresa, ambos objetos son frutas por lo tanto están basados en la clase Fruta debido a que los dos objetos comparten características (propiedades) que están definidas en la clase Fruta. Ahora pasemos este ejemplo a código.

```
1  class Fruta {} //Declaración de una clase vacía
2
3  //Crear una instancia de un objeto basado en la clase
4  const Manzana = new Fruta();
5  console.log(typeof(Manzana))
```

Lo cual nos devuelve en consola que Manzana es un objeto, como se ve a continuación.

CONSOLE X

object

Ahora, las clases pueden tener a su vez atributos y métodos. Los atributos a grandes rasgos son variables dentro de las clases y los métodos son funciones dentro de las clases. Como se ve a continuación.

```
1  class Animal{ //Definición de la clase animal
2    //Constructor: Es el método que se ejecuta al instanciarse (crearse) la clase
3    constructor(nombre, genero){
4      //Atributos
5      this.nombre=nombre;
6      this.genero=genero;
7    }
8
9    saludar(){ //Método
10     console.log(`Hola, mi nombre es ${this.nombre}`)
11   }
12 }
13
14 const perro= new Animal("Tara", "Hembra") //Objeto
15 console.log(perro)
16 perro.saludar() //Llamado al método
```

CONSOLE X

▼ Animal {nombre: "Tara", genero: "Hembra"}  
 nombre: "Tara"  
 genero: "Hembra"

Hola, mi nombre es Tara

## Herencia

Se puede denominar herencia en desarrollo de software a la característica del programa en donde una clase hija hereda las propiedades y métodos de una clase padre. Para entender mejor, por favor vea el siguiente ejemplo:

```

1  class Animal{ //Definicion de la clase animal
2  | constructor(nombre, genero){ //Construtor
3  |   //Atributos
4  |   this.nombre=nombre;
5  |   this.genero=genero;
6  | }
7  |
8  | saludar(){ //Metodo
9  | console.log(`Hola, mi nombre es ${this.nombre}`)
10 | }
11 | }
12 |
13 | class Gato extends Animal{ //Herencia
14 |   constructor(nombre, genero, color){
15 |     //Super es un metodo que llama el constructor de la clase padre
16 |     super(nombre, genero)
17 |     this.color=color;
18 |   }
19 |   //Sobreescritura
20 |   saludar(){
21 |     console.log(`Hola! Soy un gatoy mi nombre es ${this.nombre}`)
22 |   }
23 | }
24 |
25 | const milla= new Gato("Milla", "Hembra", "Gris")
26 |
27 | console.log(milla);
28 | milla.saludar();

```

CONSOLE X

▼ Gato {nombre: "Milla", genero: "Hembra", ...}  
 nombre: "Milla"  
 genero: "Hembra"  
 color: "Gris"

Hola! Soy un gato y mi nombre es Milla

## Métodos setters y getters.

En pocas palabras, son métodos que te permiten acceder a datos de los objetos, para leerlos o asignar nuevos valores. El setter lo que hace es asignar un valor y el getter se encarga de recibir un valor. Para entender mejor por favor revisar el siguiente ejemplo.

```

1  class Animal{ //Definición de la clase
2      //Constructor
3      constructor(nombre, genero){
4          //Atributos
5          this.nombre=nombre;
6          this.genero=genero;
7      }
8      saludar(){ //Metodo
9          console.log('Hola, mi nombre es ${this.nombre}')
10     }
11 }
12
13 class Gato extends Animal{
14     constructor(nombre, genero, color){
15         super(nombre, genero)
16         this.color=color;
17         this.raza= null; //se crea nulo para luego pedirlo con el setter
18     }
19     saludar(){
20         console.log('Hola! Soy un perro y mi nombre es ${this.nombre} y mi raza es $
21             {this.getRaza}')
22     }
23     get getRaza(){ //Metodo getter
24         return this.raza;
25     }
26
27     set setRaza(raza){ //Método setter
28         this.raza=raza;
29     }
30 }
31
32 const milla= new Gato("Milla", "Hembra", "Gris")
33
34 console.log(milla);
35
36 milla.setRaza = "Angora turco" //Setter
37 milla.saludar();

```

Lo que nos da en consola lo siguiente.

```

> Gato {nombre: 'Milla', genero: 'Hembra', color: 'Gris', raza: null}
  Hola! Soy un perro y mi nombre es Milla y mi raza es Angora turco

```

## 6.6. Arreglos

Explicación detallada del concepto de arreglos en el entorno de desarrollo, se desea que el estudiante domine los métodos y propiedades asociados a los arreglos.

### ¿Qué es un arreglo?

En programación se conoce como arreglos (Arrays) a una colección de elementos o cosas. Se caracterizan por guardar información como elementos del arreglo y regresa esta información cuando se le solicita. Se representan por llaves cuadradas [] y cada elemento perteneciente al arreglo se separan por comas y cada elemento tiene asociada una posición, las posiciones de los arreglos siempre comienzan por 0 e incrementa en uno con cada elemento.

### ¿Cómo crear un arreglo en JavaScript?

- Asignando un valor de arreglo a una variable.

```
1 //Array con 3 elementos
2 const colores = ['amarillo', 'rojo', 'verde']
3 //Array vacio
4 const vacio=[]
5 //Array mixto
6 const info=["Juan", 23, ["A", "B", "C"]]
7 console.log(colores)
8 console.log(vacio)
9 console.log(info)
```

```
> (3) ['amarillo', 'rojo', 'verde']
> (0) []
> (3) ['Juan', 23, Array(3)]
```



- Usando el constructor de Arreglo (new Array()).

```
1 //Array con 3 elementos
2 const colores = new Array('amarillo', 'rojo', 'verde')
3 //Array vacio
4 const vacio=new Array()
5 //Array mixto
6 const info = new Array("Juan", 23, new Array("A", "B", "C"))
7 console.log(colores)
8 console.log(vacio)
9 console.log(info)
```

```
> (3) ['amarillo', 'rojo', 'verde']
> (0) []
> (3) ['Juan', 23, Array(3)]
```

## ¿Cómo accedo a los elementos de un arreglo?

Para acceder y traer el valor de un elemento de un arreglo se necesita usar la posición en la que está. Por ejemplo.

```
1 //Array con 3 elementos
2 const colores = ['amarillo', 'rojo', 'verde']
3 //Array vacio
4 const vacio=[]
5 //Array mixto
6 const info=["Juan", 23, ["A", "B", "C"]]
7 console.log(colores[0]) //Accede al primer elemento
8 console.log(info[1])
9 console.log(colores[2])
```

CONSOLA DE DEPURACIÓN ...

Filtro (por ejemplo, text, !exclude)

C:\Program Files\nodejs\node.exe .\arrays.js

amarillo

23

verde

## Métodos y propiedades.

Una de las propiedades más importantes de los arreglos es su longitud. Muchas veces como desarrolladores vamos a necesitar saber la longitud de los arreglos para poder realizar ciertos ejercicios. Para acceder a la longitud de estos arreglos lo podemos hacer de la siguiente manera.

```
2 const colores = ['amarillo', 'rojo', 'verde']
3 console.log(`El arreglo tiene ${colores.length} elementos`)
```

CONSOLA DE DEPURACIÓN ...

Filtro (por ejemplo, text, !exclude)

```
C:\Program Files\nodejs\node.exe .\arrays.js
El arreglo tiene 3 elementos
```

Una de las aplicaciones de la propiedad length es poder iterar los arreglos, como en el siguiente ejemplo.

```
2 const colores = ['amarillo', 'rojo', 'verde']
3
4 for(let i=0; i<colores.length; i++){
5   console.log(`Elemento ${colores[i]} en la posicion ${i}`)
6 }
```

CONSOLA DE DEPURACIÓN ...

Filtro (por ejemplo, text, !exclude)

```
C:\Program Files\nodejs\node.exe .\arrays.js
Elemento amarillo en la posicion 0
Elemento rojo en la posicion 1
Elemento verde en la posicion 2
```

Entre los métodos que tenemos de los arreglos encontramos:

- Agregar un elemento. Para añadir un elemento al final del arreglo utilizamos el método push().

```
2 const colores = ['amarillo', 'rojo', 'verde']
3 colores.push('Azul')
4
5 for(let i=0; i<colores.length; i++){
6   console.log(`Elemento ${colores[i]} en la posicion ${i}`)
7 }
```



CONSOLA DE DEPURACIÓN ...

Filtro (por ejemplo, text, !exclude)

```
C:\Program Files\nodejs\node.exe .\arrays.js
Elemento amarillo en la posición 0
Elemento rojo en la posición 1
Elemento verde en la posición 2
Elemento Azul en la posición 3
```

•Eliminar un elemento. Para eliminar solo un elemento al final del arreglo utilizamos el método `pop()`, para eliminar el primer elemento de un arreglo utilizamos el método `shift()`.

```
2  const colores = ['amarillo', 'rojo', 'verde']
3  colores.pop()
4
5  for(let i=0; i<colores.length; i++){
6  |    console.log(`Elemento ${colores[i]} en la posición ${i}`)
7  }
```

CONSOLA DE DEPURACIÓN ...

Filtro (por ejemplo, text, !exclude)

```
C:\Program Files\nodejs\node.exe .\arrays.js
Elemento amarillo en la posición 0
Elemento rojo en la posición 1
```

Ejemplo de como eliminar el primer elemento de un arreglo.

```
2  const colores = ['amarillo', 'rojo', 'verde']
3  colores.shift()
4
5  for(let i=0; i<colores.length; i++){
6  |    console.log(`Elemento ${colores[i]} en la posición ${i}`)
7  }
```

CONSOLA DE DEPURACIÓN ...

Filtro (por ejemplo, text, !exclude)

```
C:\Program Files\nodejs\node.exe .\arrays.js
Elemento rojo en la posición 0
Elemento verde en la posición 1
```

•Copiar un arreglo. Para clonar todo un arreglo podemos usar el método slice(). Como se ve en el siguiente ejemplo.

```
2  const colores = ['amarillo', 'rojo', 'verde']
3  const copiaColores = colores.slice()
4
5  console.log(copiaColores)
```

CONSOLA DE DEPURACIÓN ... Filtro (por ejemplo, text, !exclude)

C:\Program Files\nodejs\node.exe .\arrays.js

> (3) ['amarillo', 'rojo', 'verde']

Para ver más métodos de los arreglos dirijase a [https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global\\_Objects/Array](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array)

## Desestructuración de arreglos.

Es una expresión de JavaScript que permite asignar elementos de arreglos a variables, es útil para mantener el código limpio y organizado. Como se ve en el siguiente ejemplo.

```
1  const [nombre, apellido, edad] = ['Laura', 'Garcia', '58']
2  console.log(`Hola, mi nombre es ${nombre} ${apellido} y tengo ${edad}`)
3
```

CONSOLA DE DEPURACIÓN ... Filtro (por ejemplo, text, !exclude)

C:\Program Files\nodejs\node.exe .\arrays.js

Hola, mi nombre es Laura Garcia y tengo 58

## 6.7. Programación avanzada

Clases en donde se explicarán y aplicarán el manejo de errores, expresiones regulares y arrow functions. En el entorno práctico se desea que el estudiante resuelva problemas de complejidad media - avanzada aplicando lo aprendido en el curso.

## Manejo de errores.

El manejo de errores con Javascript se realiza por medio de la declaración try...catch, el try es utilizado para definir un bloque de instrucciones que deben ser ejecutadas y en caso de que se presente algún error, se ejecutan las instrucciones definidas en el bloque catch. Para entender mejor ver los siguientes ejemplos.

•Ejemplo sin errores.

```
3  try {
4      console.log('Inicio');
5      let suma=15+99;
6      console.log(`El resultado es ${suma}`);
7  } catch (err) {
8      console.log('Se ignora catch porque no hay errores');
9  }
```

CONSOLA DE DEPURACIÓN ...

Filtro (por ejemplo, text, !exclude)

C:\Program Files\nodejs\node.exe .\avanzado.js  
Inicio  
El resultado es 114

•Ejemplo con errores.

```
3  try {
4      console.log('Inicio');
5      let suma=15+99;
6      console.log(`El resultado es ${resultado}`);
7      //Variable resultado no definida
8  } catch (err) {
9      console.log('Hay un error!');
10 }
```

CONSOLA DE DEPURACIÓN ...

Filtro (por ejemplo, text, !exclude)

C:\Program Files\nodejs\node.exe .\avanzado.js  
Inicio  
Hay un error!

## Expresiones regulares.

Las expresiones regulares ayudan a encontrar texto utilizando patrones representados mediante cadenas de textos en los cuales cada símbolo tiene un significado. Las expresiones regulares permiten realizar una búsqueda en el texto de manera más sencilla, las expresiones regulares se pueden escribir entre barras (/) o con el constructor del objeto RegExp. Como se ve en el siguiente ejemplo en donde se busca que el texto cumpla las siguientes condiciones:

- Que empiece por las letras O y G.
- Que termine por la letra S.

En el siguiente código se muestra un ejemplo de como se podría realizar el ejercicio sin expresiones regulares.

```
1 //Array que recibe el texto a evaluar.
2 const palabras = ["Osos", "Arroz", "Gatos", "Guitarra", "Orca", "Cama"];
3
4 // Ejercicio sin usar expresiones regulares
5 palabras.forEach(function(palabra) {
6     const primeraLetra = palabra.at(0).toLowerCase();
7     const ultimaLetra = palabra.at(-1).toLowerCase();
8
9     if ((primeraLetra === "o" || primeraLetra === "g") && (ultimaLetra === "s")) {
10         console.log(`La palabra ${palabra} cumple las restricciones.`);
11     } else {
12         console.log(`La palabra ${palabra} no cumple las condiciones`);
13     }
14 });
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
C:\Program Files\nodejs\node.exe .\expRegulares.js
La palabra Osos cumple las restricciones.
La palabra Arroz no cumple las condiciones
La palabra Gatos cumple las restricciones.
La palabra Guitarra no cumple las condiciones
La palabra Orca no cumple las condiciones
La palabra Cama no cumple las condiciones
```

Ahora, con expresiones regulares el mismo ejercicio se podría realizar de la siguiente manera.

```
1 //Array que recibe el texto a evaluar.
2 const palabras = ["Osos", "Arroz", "Gatos", "Guitarra", "Orca", "Cama"];
3
4 // Ejercicio usando expresiones regulares
5 palabras.forEach(function(palabra) {
6     const expReg = /^(o|g).+(s)$/i;
7
8     if (expReg.test(palabra)) {
9         console.log(`La palabra ${palabra} cumple las restricciones.`);
10    } else {
11        console.log(`La palabra ${palabra} no cumple las condiciones`);
12    }
13 });
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
C:\Program Files\nodejs\node.exe .\expRegulares.js
La palabra Osos cumple las restricciones.
La palabra Arroz no cumple las condiciones
La palabra Gatos cumple las restricciones.
La palabra Guitarra no cumple las condiciones
La palabra Orca no cumple las condiciones
La palabra Cama no cumple las condiciones
```

## Arrow funciones.

Las arrow functions (funciones flecha) son una nueva forma de definir funciones. Una arrow function es una forma simplificada de escribir una función debido a que:

- Estas funciones pueden ser escritas en una sola línea de código.
- No es necesario escribir la palabra reservada function.
- No es necesario escribir la palabra reservada return.
- No necesitas escribir las llaves

Estas tienen ciertas limitaciones como:

- No se deben usar como métodos.
- No se puede utilizar como constructor.

A continuación se muestra un ejemplo de cómo se puede convertir una función común en una arrow function.

```
1 //Función común
2 function saludo (nombre) {
3   |   return `Hola! ${nombre}`;
4 };
5 console.log(saludo('Maria') );
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
C:\Program Files\nodejs\node.exe .\arrowFunctions.js
Hola! Maria
```

```
7 //Arrow function
8 let saludo = nombre => `Hola! ${nombre}`;
9 console.log( saludo('Maria') );
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
C:\Program Files\nodejs\node.exe .\arrowFunctions.js
Hola! Maria
```

Se convirtió un código de 5 líneas a solo 3 líneas, simplificando la sintaxis de la función saludo.

## 6.8. Introducción JSON

Clase teórica práctica en donde se estudiará el concepto de JSON, como está constituido y cómo manipular este formato para el intercambio de información dentro de aplicaciones web, con esto el estudiante podrá realizar programas que permitan manipular información de manera más sencilla en los entornos de desarrollo.

### ¿Qué es JSON?

JSON (JavaScript Object Notation) traducido al español Notación de Objetos de JavaScript. Es básicamente una notación que es comúnmente utilizada para el intercambio de información, sirve para la comunicación entre servicios web y los clientes que los consumen, realizando la comunicación (enviando y recibiendo) la información en formato JSON, ya que para las personas leer y codificar en este lenguaje es simple, y para las máquinas es sencillo interpretarlo y generarlo.

La sintaxis de JSON está basada originalmente en la sintaxis de JavaScript, ahí viene el nombre, pero realmente es independiente de cualquier lenguaje de programación. Este formato está basado en la forma en que se escriben los objetos en JavaScript.

## Características y ventajas

JSON es una notación que se caracteriza por:

- Ser un formato moderador de datos entre sistemas.
- Consiste en un sistema de pares (clave - valor).
- Los valores asociados a las claves pueden ser cualquier tipo de datos como cadenas de texto, caracteres, números, booleanos, arreglos e incluso otros objetos JSON.
- Ser formato flexible, ligero y fácilmente transferible a través de sistemas.
- Su simplicidad, gracias a la estructuración de la información es posible que esta notación pueda ser utilizada en cualquier tipo de lenguaje de programación.

## Sintaxis de JSON.

La información obtenida en un archivo en formato JSON debe estructurarse por medio de una colección de datos organizada por medio del sistema de pares (clave - valor) o, también puede ser una lista ordenada de solo valores deben ser una lista ordenada de valores.

La clave corresponde al nombre del identificador del valor, esta clave debe ser un texto delimitado por comillas. El valor corresponde al contenido de la clave, este puede ser cualquier tipo de datos como cadenas de texto, caracteres, números, booleanos, arreglos e incluso otros objetos JSON.

Los datos almacenados en el formato JSON se componen de los siguientes elementos básicos:

- **Llaves { }** Se utilizan para delimitar el inicio y fin de los objetos.
- **Corchetes [ ]** Indican un arreglo.
- **Dos puntos :** Separan la clave con su valor correspondiente.
- **Coma ,** separan los elementos.

Para comprender mejor, vea el siguiente ejemplo.



```

1  {
2    "clave1": "valor1",
3    "clave2": null,
4    "clave3": 9,
5    "clave4": true,
6    "colores": ["Rosado", "Naranja", "Rojo", "Azul"],
7    "Juan": {
8      "telefono": "123456789",
9      "correo": "juan@correo.com",
10     "direccion": "Calle 123 # 45 - 67"
11   }
12 }

```

En Javascript se tienen métodos que ayudan a la tarea de pasar información desde JavaScript al formato JSON y viceversa. Los métodos más importantes son:

**.JSON.parse(str).** Convierte el JSON a un objeto y lo devuelve.

```

1  const json = `{
2    "nombre": "Mariana",
3    "edad": 36
4  }`;
5
6  //Pasar de JSON a Objeto
7  const usuario = JSON.parse(json);
8
9  console.log(`Hola! Soy ${usuario.nombre} y tengo ${usuario.edad}`)

```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

C:\Program Files\nodejs\node.exe .\JSON.js  
Hola! Soy Mariana y tengo 36

**.JSON.stringify(obj).** Convierte un objeto Javascript a JSON y la devuelve.

```

11  const usuario = {
12    nombre: "Mariana",
13    edad: 36,
14    amigos: ["Juan", "Fernanda", "José"]
15  };
16
17  const json = JSON.stringify(usuario);
18  console.log(json)

```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

C:\Program Files\nodejs\node.exe .\JSON.js  
{ "nombre": "Mariana", "edad": 36, "amigos": ["Juan", "Fernanda", "José"] }



## 6.9. DOM

Explicación teórico práctica de los conceptos asociados al DOM (Document Object Model), en donde se perfeccionaron las técnicas de maquetación HTML y a su vez esta maquetación se hará dinámica utilizando lo visto con JavaScript. Se espera que el estudiante logre estructurar los elementos principales que componen un aplicativo web.

### ¿Qué es DOM?

DOM es una forma de representar la página web de una manera jerárquica estructurada para que sea más fácil para los programadores y usuarios navegar por el documento. Con DOM, podemos acceder y manipular fácilmente etiquetas, ID, clases, atributos o elementos de HTML utilizando comandos o métodos proporcionados por el objeto del documento. Usando DOM, JavaScript obtiene acceso a HTML y CSS de la página web y también puede agregar comportamiento a los elementos HTML.

### Árbol de nodos del DOM

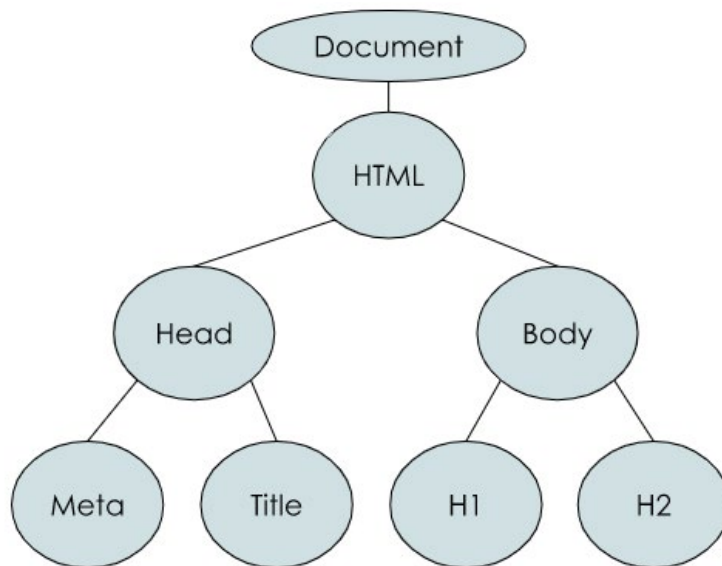
El DOM representa el documento HTML como un árbol de tres nodos. Para entender mejor la estructura del árbol primero revisemos el siguiente código.

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>DOM</title>
8  </head>
9  <body>
10     <h1>DOM</h1>
11     <h2>Estructura del DOM</h2>
12 </body>
13 </html>

```

El anterior código se puede representar mediante el árbol de la siguiente manera.



El documento HTML se llama nodo raíz, este nodo raíz tiene un nodo hijo el cual es la etiqueta `<HTML>`, que tiene a su vez dos hijos los cuales son los elementos `<head>` y `<body>`, los cuales también tienen sus propios hijos.

## Acceso al DOM con JavaScript.

Para acceder a los elementos del documento desde JavaScript existen distintos métodos que ayudan a realizar esta tarea. Los métodos principales son:

•**getElementById()**. Los id son identificadores únicos para los elementos en HTML. Esto quiere decir que no se puede tener el mismo id en distintos elementos. Por ejemplo:

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta http-equiv="X-UA-Compatible" content="IE=edge">
6    <meta name="viewport" content="width=device-width, initial-scale=1.0">
7    <title>DOM</title>
8  </head>
9  <body>
10   <h1>DOM</h1>
11   <p id="texto1">Este es mi primer texto en HTML.</p>
12   <p id="texto2">Este es mi segundo texto en HTML</p>
13 </body>
14 </html>
  
```

Para acceder al valor de los id de HTML desde JavaScript lo podemos hacer utilizando el método `getElementById()` como se ve en el siguiente ejemplo.

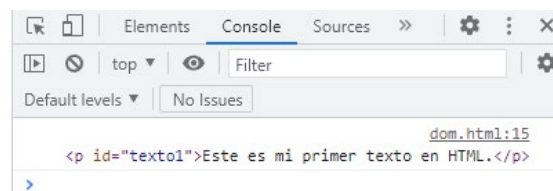
```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>DOM</title>
8 </head>
9 <body>
10  <h1>DOM</h1>
11  <p id="texto1">Este es mi primer texto en HTML.</p>
12  <p id="texto2">Este es mi segundo texto en HTML.</p>
13  <script>
14    const texto = document.getElementById("texto1");
15    console.log(texto)
16  </script>
17 </body>
18 </html>
```

Lo que nos da como resultado en consola lo siguiente.

## DOM

Este es mi primer texto en HTML.

Este es mi segundo texto en HTML



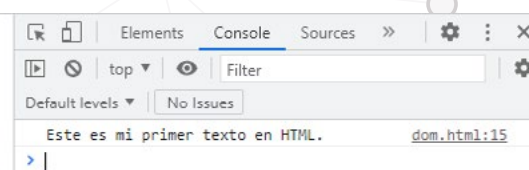
Para leer únicamente el contenido del id, en este caso, el contenido del texto entonces se puede utilizar la propiedad `textContent` como se ve a continuación.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device width, initial scale=1.0">
7   <title>DOM</title>
8 </head>
9 <body>
10  <h1>DOM</h1>
11  <p id="texto1">Este es mi primer texto en HTML.</p>
12  <p id="texto2">Este es mi segundo texto en HTML.</p>
13  <script>
14    const texto = document.getElementById("texto1");
15    console.log(texto.textContent)
16  </script>
17 </body>
18 </html>
```

## DOM

Este es mi primer texto en HTML.

Este es mi segundo texto en HTML

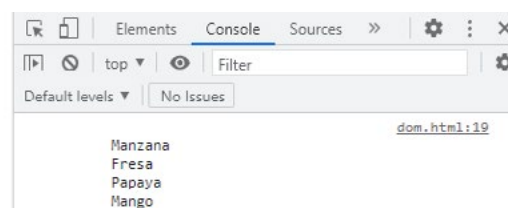


•**querySelector()**. Este método se puede utilizar para encontrar uno o más elementos.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>DOM</title>
8 </head>
9 <body>
10  <h1>Frutas</h1>
11  <ul class="list">
12    <li>Manzana</li>
13    <li>Fresa</li>
14    <li>Papaya</li>
15    <li>Mango</li>
16  </ul>
17  <script>
18    const texto = document.querySelector(".list")
19    console.log(texto.textContent)
20  </script>
21 </body>
22 </html>
```

## Frutas

- Manzana
- Fresa
- Papaya
- Mango



## 6.10. Introducción AJAX

El estudiante aprenderá a implementar AJAX en sus aplicativos web, lo cual le permitirá poner en práctica conjuntamente varias tecnologías existentes, tales como HTML o XHTML, CSS, JavaScript, DOM, XML, XSLT y XMLHttpRequest. Así sus aplicaciones web serán más dinámicas y se comprenderá de mejor manera el funcionamiento de distintos aplicativos web en general.

### ¿Qué es AJAX?

Durante la carga de un sitio web, se hacen peticiones HTTP desde el cliente hacia el servidor para solicitar información o archivos necesarios para el sitio web. AJAX (Asynchronous JavaScript And XML) es una técnica que permite realizar la petición HTTP desde Javascript para obtener/añadir/modificar/borrar información de manera asíncrona (puede comunicarse con el servidor, intercambiar datos y actualizar la página sin tener que recargar el navegador).

Cuando se realizan peticiones con AJAX se dispone de los siguientes métodos para realizar la comunicación con el servidor.

- GET. Leer.
- POST. Crear.
- PUT. Actualizar.
- DELETE. Borrar.

AJAX se trata de varias tecnologías independientes que se unen, las cuales son:

- HTML y CSS. Con estas tecnologías se realiza la presentación del sitio web.
- DOM. Con este modelo se manipula la presentación del sitio web.
- HTML, XML y JSON. Tecnologías para la manipulación de información.
- XMLHttpRequest o Fetch. Métodos para el intercambio de información de manera asíncrona.
- JavaScript. Une todas las tecnologías por medio de código.

Existen varias formas de realizar peticiones HTTP mediante AJAX, pero las principales suelen ser XMLHttpRequest y fetch.

## ¿Qué es una API?

Se conoce como API (Application Programming Interface o Interfaz de Programación de Aplicaciones) a las funciones que básicamente permiten a las aplicaciones integrar distintos sistemas con el fin de reutilizar funciones de otras aplicaciones.

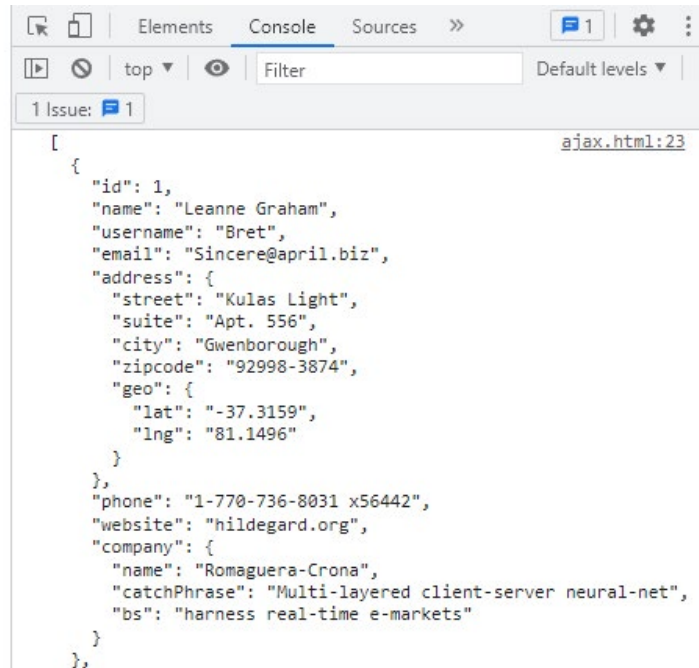
### Ejemplo.

```

1  <!DOCTYPE html>
2  <html lang="en">
3  > <head> ...
4  </head>
5  <body>
6  <h1>AJAX</h1>
7  <script>
8    function ajax() {
9      // Creación de la petición XMLHttpRequest
10     var xhr = new XMLHttpRequest();
11
12     // Haciendo la conexión
13     var url = 'https://jsonplaceholder.typicode.com/users';
14     xhr.open("GET", url, true);
15
16     // Ejecución
17     xhr.onreadystatechange = function () {
18       if (this.readyState == 4 && this.status == 200) {
19         console.log(this.responseText);
20       }
21     }
22     // Enviando la petición
23     xhr.send();
24   }
25   ajax();
26 </script>
27 </body>
28 </html>

```

## AJAX



```
[
  {
    "id": 1,
    "name": "Leanne Graham",
    "username": "Bret",
    "email": "Sincere@april.biz",
    "address": {
      "street": "Kulas Light",
      "suite": "Apt. 556",
      "city": "Gwenborough",
      "zipcode": "92998-3874",
      "geo": {
        "lat": "-37.3159",
        "lng": "81.1496"
      }
    },
    "phone": "1-770-736-8031 x56442",
    "website": "hildegard.org",
    "company": {
      "name": "Romaguera-Crona",
      "catchPhrase": "Multi-layered client-server neural-net",
      "bs": "harness real-time e-markets"
    }
  }
],
```

## 6.11. Introducción a JQuery

En esta clase el estudiante aprenderá a manejar la biblioteca JQuery en sus proyectos, también aprenderá sus distintas funciones y así, se espera simplificar muchas tareas complejas en JavaScript tales como DOM y AJAX.

### ¿Qué es jQuery?

jQuery es una librería de JavaScript, la cual no es más que un conjunto de funciones y métodos de JavaScript y gracias a esta se facilita el trabajo del desarrollo de aplicaciones. Esta librería es útil para:

- El tratamiento de los objetos de un documento HTML.
- La gestión de eventos.
- Las animaciones web.
- Interacciones AJAX.



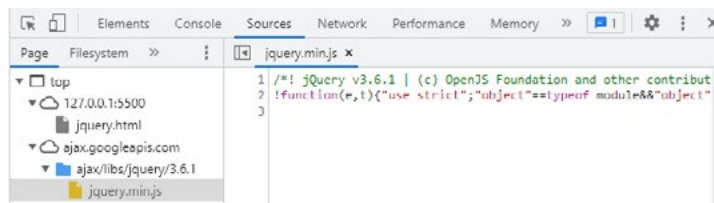
## Ejemplo

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <!-- Librería jQuery -->
8      <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.1/jquery.min.js"></script>
9      <title>jQuery</title>
10 </head>
11 <body>
12     <p>Primer parrafo</p>
13     <p>Segundo parrafo</p>
14     <p>Tercer parrafo</p>
15     <script>
16         //Oculta el parrafo cada vez que se le da click.
17         $(document).ready(function(){
18             $("p").click(function(){
19                 $(this).hide();
20             });
21         });
22     </script>
23 </body>
24 </html>

```

Primer parrafo  
Segundo parrafo  
Tercer parrafo



## 6.12. Introducción REST-API

Clase introductoria en donde se explicarán el concepto de API (Application Programming Interface), y de REST (Representational State Transfer), con esto se desea que el estudiante sea capaz de entender y poder aplicar estos conceptos en el desarrollo de aplicativos web.

### ¿Qué es una API REST?

Se denomina a API REST como el puente de comunicación entre Front End y Back End. El Back End se encarga de procesar las solicitudes y así poder consultarlas en la base de datos, mientras que el Front End responde a las interacciones que hace el usuario en el sitio web. Por ejemplo, cuando se busca algo en el navegador se suelen escribir palabras claves sobre la búsqueda que se quiere realizar, al dar enter o buscar, se obtiene un listado de resultados, así básicamente funcionan las API REST.

## Características básicas de REST.

- Sin estado (stateless).** La información del cliente no se almacena en el servidor, en cambio se almacena en el lado del cliente.
- Cliente / Servidor.** Separa las responsabilidades del cliente (Front End) y el servidor (Back End) operando de forma independiente.
- Caché.** Alguna información del lado del servidor se puede guardar en el cliente, para mejorar el rendimiento del sitio web.
- Métodos HTTP. REST** utiliza métodos como get, put, delete y post con el fin de realizar ciertas acciones en el aplicativo.

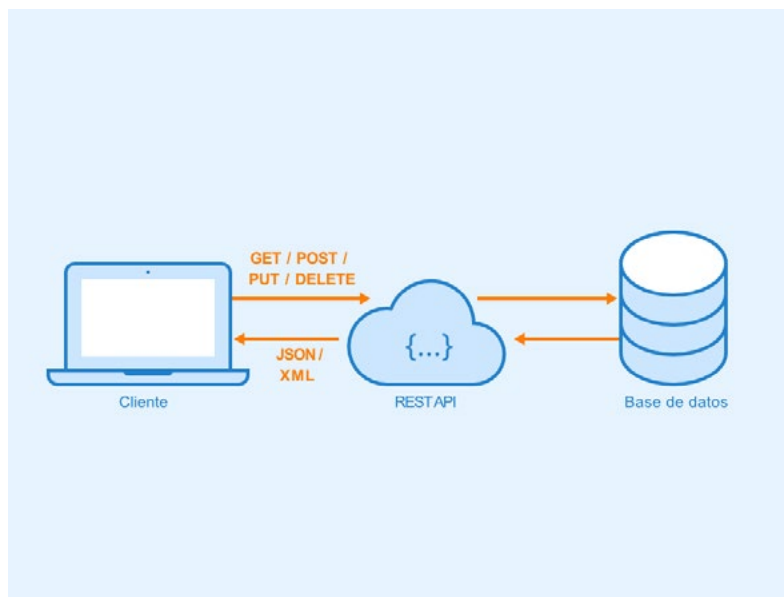


Figura 5. REST API. Fuente: <https://www.seobility.net/es/wiki/API>

## Métodos HTTP.

- GET.** Método utilizado para recuperar información. Devuelve una representación en XML o JSON y en código HTTP la respuesta OK (200) si todo sale bien o un error 404 o 400.
- POST.** Método utilizado para crear nuevos recursos. Devuelve un estado HTTP 201 si todo sale bien en la ejecución.
- PUT.** Método utilizado para actualizar o crear recursos. Devuelve un estado HTTP 200 si fue exitosa la actualización, o, un estado 201 si se crea con éxito el recurso.
- DELETE.** Método utilizado para borrar recursos.



## ¿Qué es FETCH API?

Fetch API, es una manera fácil y sencilla para obtener recursos a través de la red de forma asíncrona, de manera similar a como se realiza con AJAX pero de manera mucho más sencilla. Fetch permite al desarrollador trabajar con REST APIs. Fetch funciona con “promesas” a diferencia de AJAC que funciona por medio de devoluciones de llamada.

### Ejemplo.

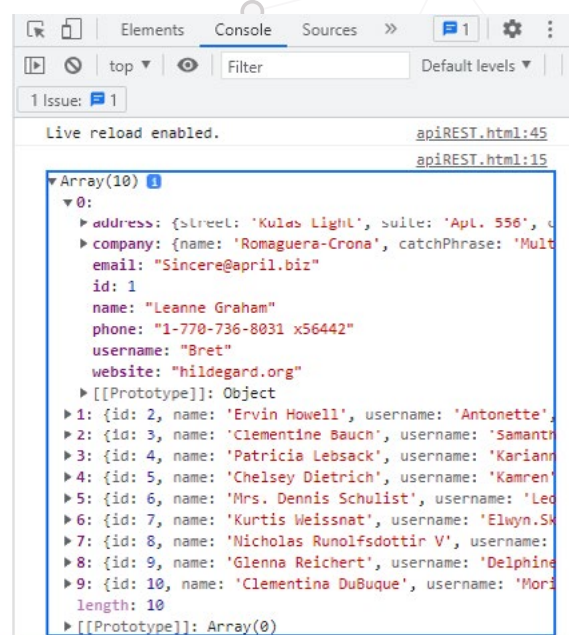
#### •GET.

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>API REST</title>
8  </head>
9  <body>
10     <h1>API REST</h1>
11     <script>
12         //GET
13         fetch('https://jsonplaceholder.typicode.com/users')
14         .then(response => response.json())
15         .then(json => console.log(json))
16     </script>
17 </body>
18 </html>

```

### API REST



## 6.13. Introducción Single-page

Clase teórica práctica en donde se explicará al estudiante el concepto de Single-page application, esto ayudará a desarrollar proyectos más ligeros, con una carga más rápida de contenidos y que facilita la depuración y corrección de errores.

### ¿Qué es Single-page application?

Se define como Single-Page Application (SPA) a los aplicativos web que ejecutan su contenido en una única página. Estos aplicativos funcionan cargando el contenido HTML, CSS y JavaScript al abrir el sitio web, al ir navegando por las páginas, módulos y secciones se carga el contenido nuevo de manera dinámica así mejorando la experiencia de usuario en la navegación. En JavaScript existe una amplia variedad de frameworks y librerías útiles para el desarrollo de estos sitios web.

En el Back End de las Single-Page Applications pueden utilizar cualquier lenguaje, pero, es importante que crean una API REST la cual se encarga de devolver el JSON para así aportar el contenido necesario para el sitio web.

## 6.14. Fundamentos de TypeScript

Desarrollo teórico práctico de TypeScript. Desde sus usos, conceptos, ventajas y desventajas en el desarrollo de aplicativos web hasta la sintaxis de esta tecnología.

### ¿Qué es TypeScript?

Typescript es un lenguaje de programación desarrollado a un nivel superior de JavaScript, TypeScript llena a JavaScript de varias características que hacen posible desarrollar aplicativos con menos errores, código más legible y coherente. Fue creado por Microsoft en 2012 con el fin de ayudar a los desarrolladores de software a realizar aplicaciones más robustas y a mayor escala. Básicamente TypeScript permite a los desarrolladores realizar un código JavaScript más limpio y seguro en las plataformas. Una de la diferencias entre JavaScript y TypeScript es que mientras que JavaScript es un lenguaje de scripting (soportado para programación a objetos), TypeScript si es un lenguaje de programación orientado a objetos. El uso de Typescript en el desarrollo de nuestras aplicaciones tiene varias ventajas importantes como:

- Detectar errores y bugs en el código JavaScript antes del tiempo de ejecución.
- Compatibilidad con las bibliotecas JS y la documentación de la API, incluyendo JQuery, BootstrapJS, React, y más.
- Utilización de NPM lo cual da acceso a más bibliotecas.
- Facilidad de leer y de auto documentarse.

## Otros materiales para profundizar

### Recursos de video



RTEA ENVIROMENT ARTIST. (2022, 10 febrero). EXPRESIONES REGULARES USO EN FORMULARIO Y CSS [Vídeo]. YouTube. [https://www.youtube.com/watch?v=-2YouzQ0BO\\_Q](https://www.youtube.com/watch?v=-2YouzQ0BO_Q)



### Material complementario

- García, I. (2022, 18 octubre). Expresiones Regulares en JavaScript. Blog SEO, Diseño Web & Gráfico | Caronte Web Studio Vitoria-Gasteiz. <https://carontestudio.com/blog/expresiones-regulares-en-javascript/>
- Tutorial gratuito sobre JavaScript - Domina las Expresiones Regulares (Regex) con Javascript. (s. f.). Udemy. <https://www.udemy.com/course/domina-las-expresiones-regulares-regex-con-javascript/>
- Tutorial gratuito sobre Seguridad informática - Escribe Código JavaScript Seguro. (s. f.). Udemy. <https://www.udemy.com/course/escribe-codigo-javascript-seguro/>
- Cv, L. (10 de marzo de 2022). Sintaxis básica SQL. #luismariocv #sql #basedatos #programming #ingenieria #datos #informatica #computacion #administracion [Vídeo] TikTok. [https://www.tiktok.com/@luismariocv/video/7150456637621308678?is\\_copy\\_url=1&is\\_from\\_webapp=v1&q=sintaxis%20basica&t=1672081378538](https://www.tiktok.com/@luismariocv/video/7150456637621308678?is_copy_url=1&is_from_webapp=v1&q=sintaxis%20basica&t=1672081378538)

### Referencias bibliográficas de la unidad



- MIREYA, S. G. (2018). ESTRUCTURAS DE DATOS.
- Gutiérrez, X. F. (1994). Estructuras de datos. Especificación, Diseño e Implementación. Edicions Universitat Politècnica Catalunya.
- Arias, Á. (2015). Aprende a programar Ajax y jQuery. Ángel Arias.



**ALCALDÍA MAYOR  
DE BOGOTÁ D.C.**  
SECRETARÍA DE EDUCACIÓN



**ATENEA**  
AGENCIA DISTRITAL PARA LA EDUCACIÓN  
SUPERIOR LA CIENCIA Y LA TECNOLOGÍA



**UNIVERSIDAD DISTRITAL  
FRANCISCO JOSÉ DE CALDAS**  
Acreditación Institucional de Alta Calidad