

Fundamentos de JavaScript

**Breve descripción:**

En este componente se abordarán los conceptos fundamentales y básicos del lenguaje de programación JavaScript en conjunto con tecnologías como HTML5, CSS3 y “frameworks” para el desarrollo de aplicaciones del lado del cliente y servidor.

**Octubre 2023**

**Tabla de contenido**

[Introducción 4](#_Toc149035167)

[1. Fundamentos de JavaScript 5](#_Toc149035168)

[1.1. Sintaxis y variables 7](#_Toc149035169)

[1.2. Declaraciones 14](#_Toc149035170)

[1.3. Operadores y objetos 17](#_Toc149035171)

[2. POO y estructuras de control 25](#_Toc149035172)

[2.1. Condicionales y clases 28](#_Toc149035173)

[2.2. Intervalos y retardos 47](#_Toc149035174)

[2.3. Nombramiento y funcionamiento 50](#_Toc149035175)

[2.4. Ciclos 51](#_Toc149035176)

[2.5. Bloques de código 56](#_Toc149035177)

[2.6. Prototipos 58](#_Toc149035178)

[2.7. Funciones y declaración de funciones 59](#_Toc149035179)

[3. Tecnologías para el desarrollo Node.js 61](#_Toc149035180)

[3.1. “Frameworks” “back-end” 61](#_Toc149035181)

[3.2. Introducción al “back-end” 62](#_Toc149035182)

[3.3. Node.Js 63](#_Toc149035183)

[3.4. Express 64](#_Toc149035184)

[3.5. Ciclos de vida y beneficios 65](#_Toc149035185)

[4. Características y bibliotecas 66](#_Toc149035186)

[4.1. Conceptos TCP 68](#_Toc149035187)

[Síntesis 79](#_Toc149035188)

[Material complementario 80](#_Toc149035189)

[Glosario 81](#_Toc149035190)

[Referencias bibliográficas 82](#_Toc149035191)

[Créditos 84](#_Toc149035192)

Introducción

En el desarrollo de este componente formativo se hace necesario hacer un recorrido por algunos elementos a tener en cuenta, para ello se debe revisar.

1. Fundamentos de JavaScript



[**Enlace de reproducción del video**](https://www.youtube.com/watch?v=f8W7riu6HiU)

|  |
| --- |
| **Síntesis del video: Fundamentos de JavaScript** |
| El desarrollo web es de suma importancia, dado que es uno de los medios más consumidos por los usuarios en la actualidad, es por esta razón que JavaScript como lenguaje de programación del lado “front-end o back-end” ha tomado gran importancia algunas de las aplicaciones más comunes desarrolladas con las tecnologías más recientes son las redes sociales como Facebook y Twitter, y aplicaciones hibridas, APIs tipo REST o SOAP, aplicaciones de “Smart Tvs”, entre muchas otras.  El manejar un solo lenguaje para el desarrollo de los dos frentes permite que los desarrolladores sean más productivos y que enfoquen su proceso de aprendizaje y de profundización en una sola tecnología, aprovechando de esta manera el performance personal y el de las herramientas. |

# Fundamentos de JavaScript

Para reconocer cuáles son los fundamentos de JavaScript, se parte de lo siguiente.

1. Video Fundamentos de JavaScript



[**Enlace de reproducción del video**](https://youtu.be/m01LAOzOiK)

|  |
| --- |
| **Síntesis del video: Fundamentos de JavaScript** |
| JavaScript es un lenguaje de programación, interpretado y creado fundamente para el control de comportamiento en aplicaciones web del lado cliente, es decir que se renderizan en el navegador del usuario, sus características es que es un lenguaje tipo “script” el cual se embebe en un documento HTML y para añadir funcionalidad interactiva a los documentos web.  Es un potente y robusto lenguaje de programación, creado por Brendan Eich, cofundador del proyecto Mozilla; con este lenguaje se pueden hacer muchas cosas, desde cosas muy básicas como carruseles de “slides”, galerías de imágenes, hasta crear efectos “parallax”, juegos en 2d, elementos en gráficos en 3d, aplicaciones reactivas con conectividad a servidores entre otros.  Es de resaltar que es un lenguaje no tipado, es decir que sus identificadores de memoria no se aparca para tipos de datos en específico, si no por el contrario cualquier tipo de dato se puede almacenar, esto hace que sea muy flexible; diferentes empresas y desarrolladores han creado componentes y herramientas en una capa superior del “core” de JavaScript, facilitando de esta manera el esfuerzo necesario para implementar ciertas cosas que son comunes dentro del desarrollo del “software”, por decirlo así, facilitan el desarrollo de funcionalidades de funcionales o componentes que son un patrón.  En sus inicios solo operaba para ejecutarse del lado cliente, pero actualmente con Node.Js se puede ejecutar del lado del servidor. También podemos encontrar super funcionalidades de la mano de TypeScript el cual es un superconjunto de JavaScript y que agrega tipado al lenguaje. |

## Sintaxis y variables

JavaScript es un lenguaje no tipado orientado a prototipos, por lo cual su sintaxis es relativamente sencilla, que por lo general es funcional.

Para poder ejecutar algún algoritmo o funcionalidad en JavaScript será necesario utilizar un documento HTML que se renderice en el navegador tal como se especifica en la figura 1.

1. Documento html con “script” de JavaScript

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

document.write("hola mundo");

</script>

</body>

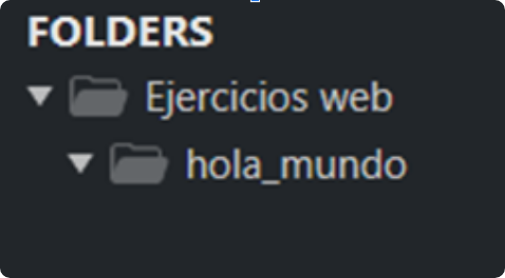
</html>

Se debe seguir las siguientes instrucciones:

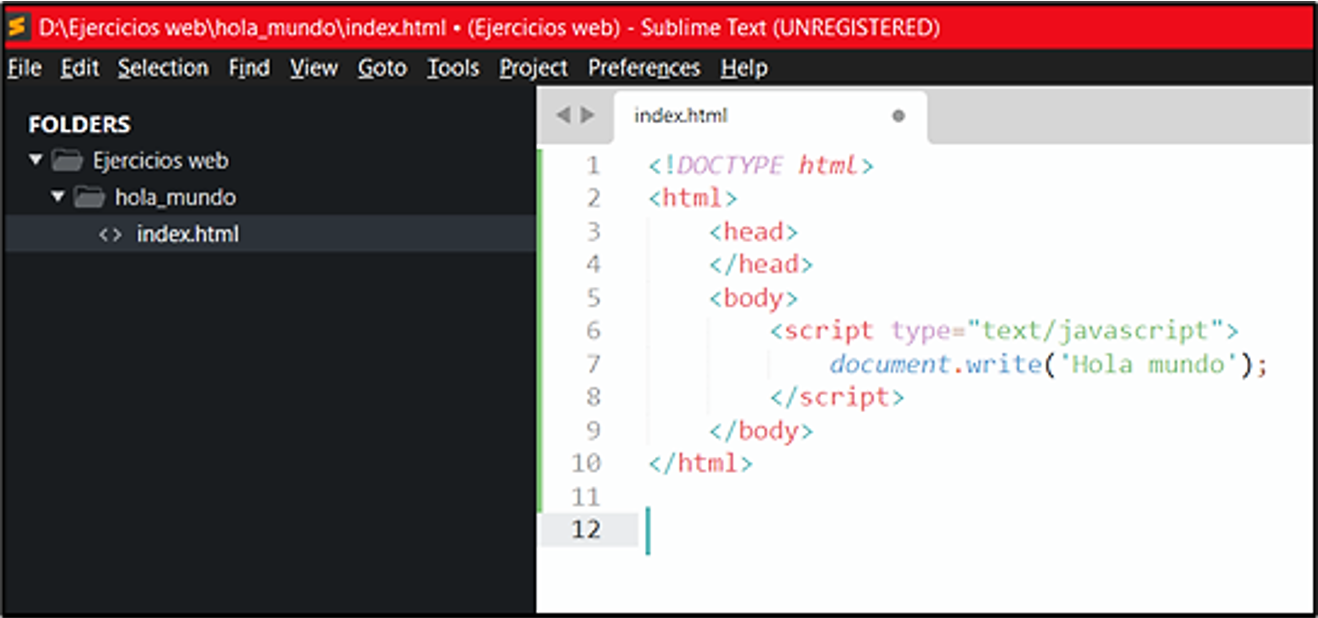
1. Cree un espacio de trabajo en su disco duro, es decir cree una carpeta con el siguiente nombre ‘Ejercicios web’.



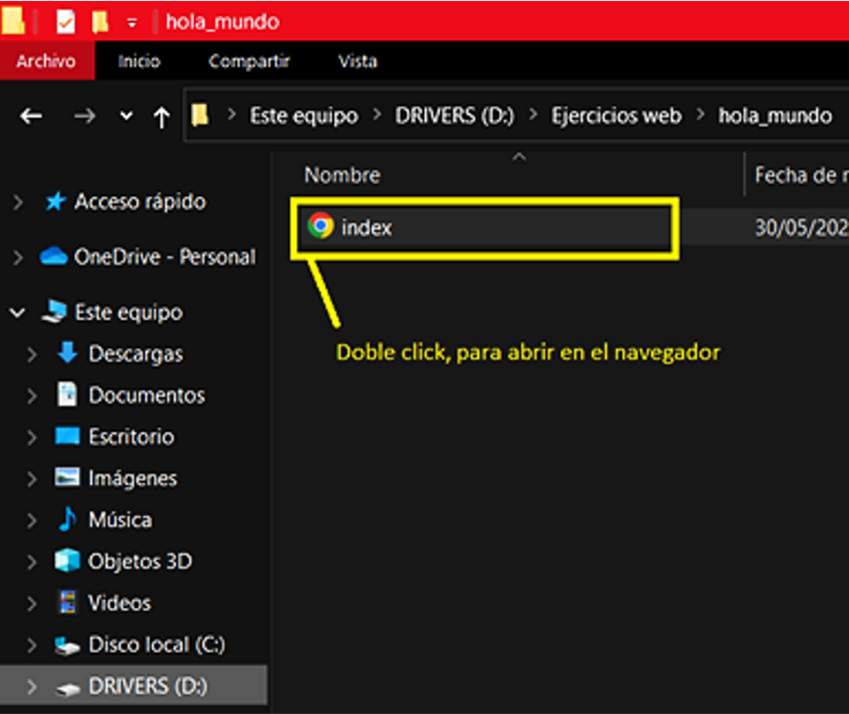
1. Por cada ejercicio se recomienda crear una carpeta dentro del espacio de trabajo, para este primer código podría denominarse ‘hola\_mundo’, por lo cual la estructura en su disco duro debería quedar de la siguiente manera:
2. Estructura de almacenamiento



1. Cree un archivo con el nombre ‘index.html’, es importante que el archivo sea de extensión html, este tipo de archivo es el que se renderiza en los navegadores web. Para esto es importante tener un IDE o un editor de código. Como recomendación puede hacer uso del editor Visual Studio Code o Sublime Text.
2. Pantallazo de Sublime Text y código HTML



1. Para ejecutar este ejemplo, puede buscar el archivo ‘index.html’ en el explorador de archivos y dar doble clic, se procederá a ejecutarse el navegador por defecto.
2. Archivo html desde explorador de archivos



1. Se sugiere ir realizando cada uno de los ejercicios y ejemplos expuestos, para la anterior imagen cree un archivo con la extensión .html y digite el código expuesto, posteriormente ejecute el archivo en algún navegador de su preferencia para que pueda evidenciar la funcionalidad.
2. Ejecución de archivo html en navegador por defecto



Además, visualiza la información que amplia de la sintaxis y variables.

**[Enlace del video](https://www.youtube.com/embed/Tk8gEEmh3Xo?si=LWW-_SMmSVBhXU7n)**

**Variables**

Son identificadores de memoria o contenedores que permiten almacenar valores, para poder hacer uso de estos en el lenguaje de JavaScript primero se deben declarar y asignar un nombre, para posteriormente hacer uso de estos

Los identificadores hacen referencia a las constantes o variables que son requeridas para el control de un flujo. Como se mencionó previamente JavaScript es un lenguaje no tipado, por lo cual para realizar una declaración de identificador de memoria este no estará atado a un tipo de dato en específico, tal como sucede en otro lenguaje altamente tipados como Java, C, C++, C#.

Por el contrario, al realizar una declaración solo se establece el espacio de memoria y al realizar la asignación automáticamente ese identificador tomará la especificación del dato asignado.

Por ejemplo, para poder crear un identificador será necesario utilizar la palabra reservada ‘var’ o ‘let’, a continuación, se define el nombre del identificador, el cual no puede ser números, no debe tener caracteres especiales y podrá ser una cadena secuencial de caracteres acompañados de números al final, ver figura 6.

1. Declaración de identificador de memoria

6 var identificador;

7 var identificador2 = "value";

8 var nombre1 = "Juan";

9 var edad = 123;

10 var casado = true;

Los datos de una variable pueden estar quemados en el código fuente tal como los expuestos en la figura 6, o se pueden recibir de la interacción con el usuario por medio de elementos de control de entrada, como la figura 7.

1. Ingreso de datos por ventanas emergentes

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

Let nombre = prompt("ingrese nombre: ", "");

Let edad = prompt("ingrese edad: ", "");

document.write("hola ");

document.write(nombre);

document.write(" , wow tienes ");

document.write(edad + " años");

</script>

</body>

</html>

En JavaScript existen diferentes tipos de datos tal como se enseña a continuación:

* **“String”**

**let miCadena = 'Bob';**

Es el tipo de dato que abarca las cadenas de texto o cadena de caracteres. Se pueden encapsular en comillas simples o sencillas, no hay distinción en el lenguaje, dada la naturaleza del lenguaje al no ser tipado.

* **Number**

**let miNumero = 10;**

Es el tipo de dato numérico y abarca cualquier número en la regla numérica, natural, entero, real. Este se expresa sin comillas.

* **Boolean**

**let miLogico = true;**

Es el tipo de dato lógico, el cual puede albergar solo dos valores true o false, palabras que son reservadas por el lenguaje y se expresan sin comillas.

* **Array**

**let miColeccion = [1,'Juan','Lopez',9, true];**

**Para acceder a cada sub posición de la colección se realiza de la siguiente manera: miColeccion[0], miColeccion[1], etc.**

Es el tipo de dato tipo colección, la cual puede almacenar varios datos en una misma estructura, ya sea un vector o una matriz.

No importa si los datos de cada posición son de diferente tipo.

* **Object**

**let miObject = document.querySelector('h1');**

**let miObject = {"name":"Juan"}**

**Aplica para todo lo anterior.**

Es el tipo de dato en el que cualquier cosa se puede almacenar, básicamente en JavaScript, todo es un object.

Las variables son requeridas dado que con estas se pueden hacer los procesos de programación dinámicos, permitiendo permutar la información a la medida que se ejecutan las funcionalidades implementadas.

## Declaraciones

En el desarrollo de “software” y en el aprendizaje de un lenguaje de programación hay diferentes aspectos a tener en cuenta, entre los cuales está la gramática y sintaxis, para obtener el máximo rendimiento al momento de realizar desarrollos e implementaciones.

**Conceptos básicos**

Entre los conceptos básicos, se puede resaltar que JavaScript es un lenguaje que no tiene distinción entre mayúsculas y minúsculas, por lo cual, si se realiza la definición de una variable en mayúsculas o minúsculas esto hará referencia al mismo identificador de memoria, en otras palabras y dicho de forma técnica es un lenguaje **case-sensitive**. También es importante recordar que se debe hacer uso de un sistema de caracteres **Unicode** por lo que no es correcto utilizar caracteres especiales para definir identificaciones de memoria.

Entre otras de las características básicas de JavaScript, es que al finalizar una línea de código o una sentencia no es necesario poner al final un ‘;’ pero en las reglas definidas por el lenguaje se sugiere que se agregue como una buena práctica.

1. Consideraciones básicas de JavaScript

1 <! DOCTYPE html>

2 <html>

3 <head>

4 </head>

5 <body>

6 <script type="text/JavaScript">

7 Let primerNombre = "juan" ;

8 primernombre = "Alex";

9 Let cantidadNiños = 13;

10 </script>

11 </body>

12 </html>

En la figura, se puede observar que en la línea 7 y 8, al definir el identificador **primerNombre** y **primernombre**, respectivamente, hacer referencia al mismo espacio en memoria, por lo cual, al ser un lenguaje con case-sensitive, inicialmente el identificador tendrá el valor de ‘Juan’ y luego será remplazado por ‘Alex’; ahora en la línea 9 se puede observar que la variable definida como **cantidadNiños**, es incorrecta dado que utiliza un carácter especial perteneciente al lenguaje español que es la ‘ñ’, esta variable deberá ser renombrada de otra forma y por último se puede observar que las líneas 7, 8 y 9, que se finaliza con ‘;’ recordando que no es una norma del lenguaje, pero es una recomendación de buena práctica.

**Comentarios**

Un comentario es un bloque de código no ejecutable, destinada para dejar anotaciones útiles para para el programador, que por lo general es usado para documentar variables, operaciones, funciones y se pueden aplicar en la mayoría de los elementos del programa. La sintaxis de los bloques de comentarios en JavaScript es igual que muchos lenguajes de programación.

1. Comentarios en JavaScript

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

// un comentario de una línea

/\*

este es un comentario

más largo, de varias líneas

\*/

/\* Sin embargo, no puedes /\* anidar comentarios \*/ SyntaxError \* /;

</script>

</body>

</html>

En la figura 9, se puede observar que hay varias formas de crear comentarios, si se requiere un comentario de una sola línea solo con iniciar la sentencia con // será suficiente, pero dado el caso que se requiera que sea un párrafo o más de una línea deberá iniciarse así /\* y cerrarse de la siguiente manera \*/. Adicional a eso recordar que no se pueden anidar comentarios.

Los conceptos básicos del lenguaje abarcan otros temas, los cuales se irán abarcando en el desarrollo de este componente.

## Operadores y objetos

Los operadores en JavaScript son signos o símbolos que sirven para definir expresiones, las cuales pueden ser de asignación, lógicas, aritméticas, de comparación y referencia.

**Operadores de asignación**

Son aquellos operadores que definen un valor del operando derecho al operando izquierdo, siempre y cuando el operando izquierdo no sea un valor literal, es decir que sea un identificador de memoria, el operando de asignación más utilizado y sencillo de recordar es el igual (=), el cual permite asignar un valor a la variable definida en el lado izquierdo, ver figura 10.

1. Operador de asignación (=)

var suma = 1 + 2;

var nota1 = 1;

var nota2 = 4;

var nota3 = nota1 + 1;

var promedio = (nota1 + nota2 + nota3) / 3

**Tabla de operadores de asignación compuestas**. Ver documento: **Tabla de operadores de asignación compuestas** en la carpeta de anexos**,** allí se encuentra la información de manera completa.

**Operadores de comparación**

Son aquellos que realizan una evaluación entre los operandos y devuelven un valor lógico, es decir un valor de verdadero o falso, los operandos pueden ser números, cadenas, booleanos, caracteres; dado el caso que los operandos no sean del mismo tipo, el intérprete de JavaScript intentará realizar una conversión implícita para hacer la evaluación, ver figura 11.

1. Operador de comparación (==)

var var1 = 3;

var var2 = 4;

var result = var1 == var2;

En la figura 11 se puede observar la creación de dos identificadores de memoria var1 y var2, los cuales tienen una asignación de valor respectivamente, posteriormente se crea un identificador denominado result, este almacena el valor de comparar si var1 es igual a var2, el cual es falso; por lo tanto, el valor de result es el valor de ‘false’, se puede observar así:

* **Operador Igual (==)**

**Descripción**

Retorna true si los operandos tienen el mismo valor.

**Uso que devuelven true**

4 == var1

"5" == var1

6 == '6'

* **Operador Diferente, desigual (!=)**

**Descripción**

Retorna true si los operandos no presentan el mismo valor.

**Uso que devuelven true**

var1 != 2

var2 != "2"

* **Operador Estrictamente igual (===)**

**Descripción**

Retorna true si los operandos son iguales y adicionalmente son del mismo tipo.

**Uso que devuelven true**

1 === var1

* **Operador Desigualdad estricta (!==)**

**Descripción**

Retorna true si los operandos son del mismo tipo pero no presentan el mismo valor, o si son de diferente tipo.

**Uso que devuelven true**

var1 !== "7"

5 !== '5'

* **Operador Mayor que (>)**

**Descripción**

Retorna true si el operando izquierdo presenta un valor mayor que el operando derecho.

**Uso que devuelven true**

var1 > var2

"13" > 4

* **Operador Mayor o igual que (>=)**

**Descripción**

Retorna true si el operando izquierdo presenta un valor mayor o igual que el operando derecho.

**Uso que devuelven true**

var1 >= var2

var1 >= 6

* **Operador Menor que (<)**

**Descripción**

Retorna true si el operando izquierdo presenta un valor menor que el operando derecho.

**Uso que devuelven true**

var1 < var4

"6" < 20

* **Operador Menor o igual (<=)**

**Descripción**

Retorna true si el operando izquierdo presenta un valor menor o igual que el operando derecho.

**Uso que devuelven true**

var1 <= var4

var2 <= 15

**Nota: (=>) no es un operador, sino la notación para Funciones flecha.**

Anteriormente se pudo observar los diferentes tipos de operadores de comparación, es importante recordar que estos ayudan a evaluar dos operandos, retornando con ello un valor booleano de verdadero o falso, esto es útil al construir una sentencia relacional como las que son usadas en los condicionales o algunos de los ciclos.

**Operadores aritméticos**

Son aquellos que realizan operaciones matemáticas, ya sea con valores literales o variables, entre estos operadores encontramos el (+) (-) (/) (\*), ver figura 12.

1. Operadores aritméticos comunes

var var1 = 3;

var var2 = 4;

var r1 = var1 + 3; // 6

var r2 = 5 - 4; // 1

var r3 = 3 \* var2; //12

var r4 = var2 / 2; // 2

var r5 = 12 % 2; // 0

Además, se debe observar la siguiente tabla:

| Operador | Descripción | Uso |
| --- | --- | --- |
| Residuo (%) | Operador binario o también denominado división modular. Retorna el resto entero de dividir los dos operandos. | 12 % 5 devuelve 2. |
| Incremento (++) | Suma uno a su operando. Si se usa como operador prefijo (++x), retorna el valor de su operando después de agregar uno; si se usa como operador sufijo (x++), retorna el valor de su operando antes de agregar uno. | Si x es 3, ++x establece x en 4 y retorna 4, mientras que x++ retorna 3 y, solo entonces, establece x en 4. |
| Decremento (--) | Resta un uno de su operando. El valor de retorno es equivalente al del operador de incremento. | Si x es 3, entonces --x establece x en 2 y retorna 2, mientras que x-- retorna 3 y, solo entonces, establece x en 2. |
| Negación unaria (-) | Retorna la negación de su operando. | Si x es 3, entonces -x retorna -3. |
| Positivo unario (+) | Busca convertir el operando en un número, si este no lo es. | +"3" retorna 3.  +true devuelve 1. |
| Operador de exponenciación (\*\*) | Calcula la base a la potencia de exponente, es decir, baseexponente | 2 \*\* 3 returns 8.  10 \*\* -1 returns 0.1. |

Estos operadores solo pueden ser usados en el ámbito matemático, operando valores numéricos, con identificadores de memoria del tipo numérico.

**Operadores lógicos**

Son aquellos operadores que se usan para establecer uniones entre valores booleanos, y generar un valor booleano. Tener en cuenta los siguientes ejemplos:

* **Ejemplo de operador && (y)**

var a1 = true && true; // t && t devuelve true

var a2 = true && false; // t && f devuelve false

var a3 = false && true; // f && t devuelve false

var a4 = false && (3 == 4); // f && f devuelve false

var a5 = 'Cat' && 'Dog'; // t && t devuelve Dog

var a6 = false && 'Cat'; // f && t devuelve false

var a7 = 'Cat' && false; // t && f devuelve false

* **Ejemplo de operador || (o)**

var o1 = true || true; // t || t devuelve true

var o2 = false || true; // f || t devuelve true

var o3 = true || false; // t || f devuelve true

var o4 = false || (3 == 4); // f || f devuelve false

var o5 = 'Cat' || 'Dog'; // t || t devuelve Cat

var o6 = false || 'Cat'; // f || t devuelve Cat

var o7 = 'Cat' || false; // t || f devuelve Cat

* **Ejemplo de operador ¡ (negación)**

var n1 = !true; // t devuelve false

var n2 = !false; // f devuelve true

var n3 = !'Cat'; // t devuelve false

Para completar la información anterior revisar en detalle los datos que se presentan entre operador, uso y la descripción de cada uno de los que se relacionan.

| Operador | Uso | Descripción |
| --- | --- | --- |
| AND Lógico (&&) | expresion1 && expresion2 | Tener en cuenta que cada expresión debe retornar un valor lógico true/false. Dado el caso que las dos expresiones sean true, el operador && devolverá true, en caso contrario siempre retornara false. |
| OR Lógico (||) | expresion1 || expresion2 | Tener en cuenta que cada expresión debe retornar un valor lógico true/false. Dado el caso que una de las dos expresiones sean true, el operador || devolverá true, en caso contrario siempre retornara false. |
| NOT Lógico (!) | ! expresion | Tener en cuenta que la expresión debe retornar un valor lógico true/false. Dado el caso que la expresión sea true, al negarse retornara el valor contrario, es decir, false; y en caso contrario ocurrirá lo mismo. |

**Objetos**

JavaScript, es un lenguaje que está construido en un paradigma basado en objetos, recordando que un objeto es una estructura que permite la colección de propiedades, atributos, características o datos, estos a su vez tienen asociado un valor es decir una información relacionada a lo que representa.

Tener en cuenta el siguiente ejemplo:

Imagen que contiene un ejemplo de objetos JavaScript, el cual tiene la siguiente información a modo de historieta:

Comencemos...tenemos el objeto tipo persona, y ese objeto tiene como atributos el nombre y la edad, esos atributos deberán almacenar información 

tales como ‘Juan’ y 18, datos que corresponden a las propiedades definidas en el objeto. 

Al igual que en la mayoría de los lenguajes de programación, en JavaScript un objeto también representa la abstracción de un objeto en la vida real.

Se muestra un código Declaración de un objeto.

Como se puede observar en la figura Declaración de un objeto, en JavaScript se pueden definir objetos realizando la instanciación de la clase Object y a la medida que se requiera agregar propiedades a la instancia, de la misma manera se podrán añadir valores a las propiedades definidas. 

Este objeto también se puede definir con el identificado de objeto tal como lo podemos observar en la figura Notación de objeto. 

Se muestra un código Notación de un objeto.


Estas son algunas de las particularidades de JavaScript para la definición de objetos en el lenguaje.

# POO y estructuras de control

La POO (Programación Orientada a Objetos) es un paradigma de programación, que pretende solucionar problemas informáticos, realizando la abstracción de las funcionalidades en clases y objetos. Es importante recordar que las clases son una estructura que establece las propiedades y comportamientos, por medio de atributos y métodos, mientras que los objetos son una instancia de lo definido en la clase, dicho de otra manera, la clase es una plantilla y el objeto es una referencia concreta tomada de la plantilla.

¿Cómo se realiza? Observar a continuación:

1. **Persona:** comencemos creando una clase con un constructor y pasando como argumento el valor de una propiedad, ver figura Función constructora en JavaScript. En este caso se crea una función constructora para la estructura la cual recibe el argumento de nombre.
2. **Saludar:** también se puede observar como el procedimiento de saludar, el cual al ser ejecutado lanza una alerta haciendo uso de la propiedad nombre.
3. **Función constructora en JavaScript:** en JavaScript, no existen constructores tal como se ven en otros lenguajes tales como Java, C#, entre otros, por el contrario, por medio de una función y el scope this, se podrá construir todas las propiedades requeridas para la acción de instanciación.

**Función constructora**

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

function Persona(nombre) {

this.nombre = function() {

alert('Hola yo soy ' + this.nombre + '.');

};

}

var persona1 = new Persona('Gabriela');

var persona2 = new Persona('Mateo');

</script>

</body>

</html>

1. **New:** sirve para indicar al intérprete del navegador que se va a crear una nueva instancia de la estructura Persona, en el caso del ejemplo anterior se están estableciendo dos objetos diferentes persona1 y persona2.

**Estructura secuencial**

Las estructuras de control permiten establecer los flujos para la ejecución de un programa o algoritmo, en el caso de que en un problema o fragmento de código solo exista la entrada, el proceso y la salida se le determina estructura secuencial, ver figura 13.

1. Estructura secuencial

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

Let numero1 = prompt ("Ingrese numero1: ", "");

Let numero2 = prompt ("Ingrese numero2: ", "");

numero1 = parseInt (numero1);

numero2 = parseInt (numero2);

Let resusltado = numero2 + numero2;

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

</script>

</body>

</html>

Las soluciones que ejecutan estructuras secuenciales suelen ser los más sencillos, dado que dependiendo de lo que se requiera solucionar se necesitarán de otras estructuras que cambiarán el flujo de control, es de resaltar que no todos los problemas, algoritmos o problemas informáticos se solucionarán con estructuras secuenciales, dado que existirán problemas más complejos, los cuales requerirán tomar decisiones, repeticiones de sentencias con la finalidad de evitar duplicación de funcionalidad, uso de bloques de código para reutilizar funcionalidades, entre otros elementos.

Observa los elementos de la estructura secuencial.

**[Enlace del video](https://www.youtube.com/embed/Ujk0Ej32Reg?si=HDO6DIzFTLSGNRWj)**

## Condicionales y clases

Dado que no todos los problemas se pueden resolver de forma secuencial, se tienen las estructuras condicionales las cuales son las encargadas de poder tomar decisiones, al igual que la vida cotidiana, en los algoritmos informáticos es necesario hacerlo.

Para tomar decisiones en procesos algorítmicos será necesario hacer uso de estructuras condicionales, que de forma resumida permiten tomar un flujo u otro en el código fuente planteado en el programa, es decir a diferencia de la estructura secuencial, en la que todo el código se ejecutaba, de tal manera que de la línea 1 hasta la línea 10 se ejecutaba de forma ordenada y consecutiva, con las condiciones no pasa esto, de acuerdo a las condiciones establecidas en las sentencias de evaluación y los valores dados a las variables de control, existirán bloques de código que no se ejecutarán. Para ello tener en cuenta lo que se comparte a continuación:

* **Estructura condicional simple, “if”**

Es la estructura condicional que permite ejecutar líneas de código o funcionalidades dependiendo de una decisión y si el resultado es afirmativo es decir verdadero, solo en ese caso se ejecutarán las acciones, por otro lado, en el condicional simple, o sea, cuando el resultado es negativo o falso, no ocurre nada.

El siguiente ejemplo hace alusión a una persona que se le indica un mensaje de aprobación de curso con una nota superior o igual a 4, ver figura Condicional simple. En caso de que la nota será inferior a ese valor no se generará mensaje.

**Condicional simple**

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

Let nombre = prompt ("Ingrese nombre: ", "");

Let nota = prompt ("Ingrese nota: ", "");

nota = parseInt (nota);

// condicional simple

if (nota > 4) {

document.write(nombre + " es sobresaliente con la nota: " + nota);

}

</script>

</body>

</html>

* **Estructura condicional simple, “if**”

La estructura condicional por la palabra reservada “**if”** seguido de paréntesis, en el cual debe ir la condición que se está evaluando, esta condición es una sentencia relacional o lógica que genera un valor booleano es decir verdadero o falso, posterior a ello están las llaves que representan el cuerpo de código o bloque de código que se ejecutara si el valor evaluado es verdadero.

**Condicional simple**

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

Let nombre = prompt ("Ingrese nombre: ", "");

Let nota = prompt ("Ingrese nota: ", "");

nota = parseInt (nota);

// condicional simple

if (nota > 4) {

document.write(nombre + " es sobresaliente con la nota: " + nota);

}

</script>

</body>

</html>

* **Condicional doble o compuesta,** **“if – else if – else”**

Es posible que en la vida cotidiana se nos presente una opción u otra, por decirlo así podemos ir por x o y camino, es importante entender que en este caso no será posible ejecutar las acciones de las dos ramas, si pasa el suceso de verdadero solo se ejecutará las acciones determinadas cuando pasa ese suceso o viceversa si es falso solo se ejecutarán las acciones definidas por esa opción.

Por ejemplo, se ingresa 2 números y se quiere saber si el primero es mayor que el segundo, o lo contrario. Ver figura Condicional doble.

**Condicional doble**

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

Let numero1 = prompt ("Ingrese número 1: ", "");

Let numero2 = prompt ("Ingrese número 2: ", "");

numero1 = parseInt (numero1);

numero2 = parseInt (numero2);

// condicional doble

if (numero1 > nuemro2) {

document.write("el mayor es: " + numero1);

} else {

document.write("el mayor es: " + numero2);

}

</script>

</body>

</html>

* **Condicional doble o compuesta, “if – else if – else”**

La estructura de un condicional compuesta al igual que la simple contiene la evaluación de verdadero “**if”**, adicional a ello se encuentra el bloque de validación alterno que se denomina “**else”**, este encapsula el cuerpo de código o las sentencias que se ejecutan si el bloque de verdadero no se cumple, es decir, si la condición es falsa se ejecuta el bloque estipulado en el “else”.

**Condicional** doble

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

Let numero1 = prompt ("Ingrese número 1: ", "");

Let numero2 = prompt ("Ingrese número 2: ", "");

numero1 = parseInt (numero1);

numero2 = parseInt (numero2);

// condicional doble

if (numero1 > nuemro2) {

document.write("el mayor es: " + numero1);

} else {

document.write("el mayor es: " + numero2);

}

</script>

</body>

</html>

* **Condicional doble o compuesta, “if – else if – else”**

Para resumir lo que se ha expuesto hasta el momento, podemos indicar que el condicional simple solamente evalúa una opción, que al ser verdadera permite ejecutar establecido en el interior del cuerpo de definición, el condicional doble, por el contrario permite ejecutar sentencias de código cuando el bloque principal no se ejecuta, en este caso cuando el valor de la condición es falsa; pero puede haber casos en los cuales, no solo se evalué de forma binaria, es decir, no solo tenga dos posibles flujos.

**Condicional doble**

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

Let numero1 = prompt ("Ingrese número 1: ", "");

Let numero2 = prompt ("Ingrese número 2: ", "");

numero1 = parseInt (numero1);

numero2 = parseInt (numero2);

// condicional doble

if (numero1 > nuemro2) {

document.write("el mayor es: " + numero1);

} else {

document.write("el mayor es: " + numero2);

}

</script>

</body>

</html>

* **Condicional doble o compuesta, “if – else if – else”**

En el ejemplo anterior, relacionado con cuál de los dos números ingresados el programa solo podrá decir que el primero o el segundo es mayor, pero el programa como tal tiene una falla, dado que hay una posibilidad adicional, la cual consiste en que puede que el numero 1 no es mayor que el número 2, y que el numero 2 es mayor al número 1, es decir que los dos números son iguales.

**Condicional doble**

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

Let numero1 = prompt ("Ingrese número 1: ", "");

Let numero2 = prompt ("Ingrese número 2: ", "");

numero1 = parseInt (numero1);

numero2 = parseInt (numero2);

// condicional doble

if (numero1 > nuemro2) {

document.write("el mayor es: " + numero1);

} else {

document.write("el mayor es: " + numero2);

}

</script>

</body>

</html>

* **Condicional doble o compuesta, “if – else if – else”**

En este caso es posible que la estructura “**if”** nos brinde el soporte para poder realizar una siguiente evaluación condicional, tal como lo veremos en la figura Condicional con múltiples condiciones. ero o el segundo es mayor, pero el programa como tal tiene una falla, dado que hay una posibilidad adicional, la cual consiste en que puede que el numero 1 no es mayor que el número 2, y que el numero 2 es mayor al número 1, es decir que los dos números son iguales.

**Condicional con múltiples condiciones**

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

var num1, num2;

num1 = prompt ("Ingrese el primer número:", " ");

num2 = prompt ("Ingrese el primer número:", " ");

num1 = parseInt (num1);

num2 = parseInt (num2);

if (num1 > num2) {

document.write("el mayor es: " + num1);

{

else if (num2 > num1) {

document.write("el mayor es: " + num2);

}

} else {

document.write("los números son iguales");

}

</script>

</body>

</html>

* **Condicional doble o compuesta, “if – else if – else”**

En este ejemplo podemos ver que se añade la estructura “else if”, el cual permite realizar una validación adicional, dado que la primera condición llegue a ser falsa, dicho así podría leerse de la siguiente manera:

**Si el numero 1 es mayor al número 2**

**El mayor es el numero 1**

**Si no, si el numero 2 es mayor al número 1**

**El mayor es el numero 2**

**Si no**

**Los dos números son iguales**

* **Condicional doble o compuesta, “if – else if – else”**

Podemos deducir que la estructura “else” llega a convertirse en el bloque por defecto, llegado el caso que la estructura condicional no tenga un condicional que se llegase a ejecutar como afirmativo, es decir si no hay una condición que se cumpla. Esta estructura permite hacer la evaluación de infinida de condiciones, sin importar la cantidad, es recomendado no presentar ambigüedades, también recordar que la ejecución de las condiciones es secuencial, si se llegase a ejecutar por decirlo así la primera condición, el flujo del algoritmo seguiría al finalizar toda la estructura condicional, es decir al finalizar el “else” sin llegar a ejecutar su lógica o bloque de código.

Ten en cuenta para las condicionales la siguiente información.

**[Enlace del video](https://www.youtube.com/embed/zefoFN81By8?si=6ztNC90wWztIaDO8)**

**Condicional anidado**

Se debe recordar que cada bloque de código encapsulado en llaves {… } es totalmente independiente a otro bloque de código, por lo cual es posible que se pueda definir una estructura condicional dentro de alguno de los flujos de otro condicional, lo importante en este tipo de casos es que cada condicional realizará la validación de condiciones que no son compatibles, es decir será otro grupo de validación, con otras características, ver figura 14.

1. Condicionales anidados

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

var edad, opcion;

edad = prompt("Ingrese edad:", "");

if (edad<18) {

opcion = prompt("Quiere helado(h) o pastel(p):", "");

if(opcion==="h") {

document.write("Bono de 1000 pesos");

} else if(opcion==="p") {

document.write("Bono de 1500 pesos");

} else {

document.write("Opción no valida");

}

}

else {

opcion = prompt("Ingrese genero(m/f/o) :", "");

if(opcion==="m") {

document.write("15% desc en camisas y camisetas");

} else if(opcion==="f") {

document.write("20% desc en faldas y 15% desc en blusas");

} else {

document.write("5% desc en cualquier prenda");

}

}

</script>

</body>

</html>

Suponiendo que se tiene una super tienda en la cual venden confitería y prendas de vestir, en el cual tienen apartados para las personas de cualquier tipo de edad, esta tienda quiere saber si quien realiza la compra en su tienda es menor o mayor de edad, dado eso ofrecerá unos beneficios de acuerdo con su edad. Dado que sean menores de edad realizará una bonificación o descuento, si el cliente es menor de edad el beneficio se aplica si se realiza la compra de pastel o helado; por otro lado, si se es mayor de edad, se requiere validar la identidad de género del cliente y con ello aplicar una serie de descuentos de acuerdo con la clasificación.

En la figura 14 se puede observar que hay un condicional que realiza la primera evaluación la cual consiste en saber si el valor de la edad es menor a 18, en el bloque de verdadero se realiza un condicional anidado lo cual evalúa si el usuario seleccionó helado o pastel; por otro lado, en el bloque del sino del condicional principal se realiza la validación de identidad de género, masculino, femenino u otro, lo cual lleva a una serie de descuentos. Si se analiza, los condicionales internos o anidados se pretende evaluar unas condiciones que no son compatibles con la condición de la estructura condicional principal y dada la naturaleza clasificatoria de la misma se requiere de las estructuras internas para realizar una segunda validación póstuma a la clasificación inicial, se puede observar cómo se presenta para estructura “switch” y las clases.

* + **Estructura “switch”**

Es una alternativa al condicional, aunque una de sus principales diferencias es que su estructura de validación no hace uso de una sentencia condicional, del tipo mayor que, menor que, etc., solo si el valor de la variable coincide con un valor en específico que por lo general es una constante o un valor literal. Por ejemplo, ver figura Estructura “switch”.

**Estructura “switch”**

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

var diaSemana;

diaSemana = prompt("Ingrese día de la semana 1-7:", "");

switch (mes) {

case 1:

document.write("Lunes");

breack;

case 2:

document.write("Martes");

breack;

case 3:

document.write("Miercoles");

breack;

case 4:

document.write("Jueves");

breack;

case 5:

document.write("Viernes");

breack;

case 6:

document.write("Sabado");

breack;

case 7:

document.write("Domingo");

breack;

default:

document.write("Día no valido");

}

</script>

</body>

</html>

* + **Estructura “switch”**

Se puede observar que a diferencia al condicional **“if”**, se usa la palabra reservada “switch”, el cual cumple una función similar, seguido de la variable que se va a evaluar, la evaluación consiste en validar si su valor corresponde a alguno de los cases definidos en el interior de la estructura. Por último, se tiene el caso default, que cumpliría una función similar al else, el cual es que, si no se aplica ninguno de los casos definidos en el bloque de control, se ejecuta ese bloque por defecto. Es recomendado tener en cuenta que los cases no son bloques independientes, por lo cual para salir de la opción utilizan la palabra reservada ***break***, si esta palabra no es usada saltara a la siguiente instrucción de los cases definidos sin realizar la validación.

**Estructura “switch”**

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

var diaSemana;

diaSemana = prompt("Ingrese día de la semana 1-7:", "");

switch (mes) {

case 1:

document.write("Lunes");

breack;

case 2:

document.write("Martes");

breack;

case 3:

document.write("Miercoles");

breack;

case 4:

document.write("Jueves");

breack;

case 5:

document.write("Viernes");

breack;

case 6:

document.write("Sabado");

breack;

case 7:

document.write("Domingo");

breack;

default:

document.write("Día no valido");

}

</script>

</body>

</html>

* + **Clase**

Fueron incluidas en 2015, con la finalidad de mejorar la sintaxis basada en la herencia orientada a prototipos, pero no representa una mejora en la misma, solo de manera estructura y la facilidad para realizar herencias.

La forma para definir una clase en JavaScript es por medio de las estructuras especializadas similares a los de otros lenguajes y es la estructura class, ver figura Modo estricto de clase en JavaScript.

**Modo estricto de clase en JavaScript**

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

class Rectangulo {

constructor (alto, ancho) {

this.alto = alto;

this.ancho = ancho;

}

//Getter

get area() {

return this.calcArea();

}

//Método

calcArea() {

return this.alto \* this.ancho;

}

}

const cuadrado = new Rectangulo(10,10);

console.log(cuadrado.area); //100

</script>

</body>

</html>

* + **Clase**

La estructura definida en la anterior figura denota el modo estricto, el cual permite mejora el rendimiento al momento de renderizarse, contiene un constructor, dos atributos, alto y ancho, y dos métodos.

Una particularidad de POO y de las clases es la herencia, y con este tipo de estructuras es posible que, desde JavaScript, se pueda definir clases que hereden la abstracción de clases padres, de la misma manera podemos observar otra característica de este paradigma y es la sobrecarga de herencia.

Para obtener una herencia, se hace uso de la palabra reservada extends, tal como A extends B, denotando de esta manera que la clase A hereda o es una clase hija de la clase B.

**Modo estricto de clase en JavaScript**

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

class Rectangulo {

constructor (alto, ancho) {

this.alto = alto;

this.ancho = ancho;

}

//Getter

get area() {

return this.calcArea();

}

//Método

calcArea() {

return this.alto \* this.ancho;

}

}

const cuadrado = new Rectangulo(10,10);

console.log(cuadrado.area); //100

</script>

</body>

</html>

Por otro lado, la sobreescritura consiste en reescribir un método, por ejemplo, haciendo la abstracción de los seres vivos, hacemos ruido, los perros ladran, los pájaros cantan, los humanos hablan, etc., si tenemos un método en la clase A tal como hacerRuido, y en la clase B un método igual hacerRuido, esto genera una sobreescritura, prevaleciendo el método de la clase más especializada, es decir, la última clase definida.

Para aclarar los puntos dichos previamente analizar y ver la figura 15.

1. Herencia y sobreescritura

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

class Animal {

constructor (nombre) {

this.nombre = nombre;

}

hablar() {

console.log(this.nombre + ' hace mucho ruido.');

}

}

class Perro extends Animal {

hablar() {

console.log(this.nombre + ' ladra.');

}

}

var animalito = new Perro("Pepito");

var perrito = new Perro("Chicky");

consol.log(animalito.hablar()); //Pepito hace mucho ruido.

consol.log(perrito.hablar()); //Chicky ladra.

</script>

</body>

</html>

De acuerdo con los temas que se han visto hasta el momento se puede analizar que hay varias formas de poder establecer estructuras para generar instancias de objetos, hasta el momento esta es la más nueva, pero a su vez es la más similar a lenguajes altamente tipados.

Observa aspectos desde la estructura “switch”.

**[Enlace del video](https://www.youtube.com/embed/bsnRjlbU5c4?si=YAGe7--o-D9VhN7o)**

## Intervalos y retardos

Los intervalos y retardos son bloques de código definidos por el lenguaje para pausar una ejecución o para repetir una instrucción en x cantidad de tiempo, tener en cuenta lo siguiente:

* + **Bloque “setTimeout”**

Es una función que está determinada para pausar o retrasar la ejecución de una funcionalidad dependiendo de la cantidad de tiempo definida, esta función recibe 2 parámetros, uno de ellos es el tiempo de ejecución y debe estar expresado en milisegundos, el otro es una función, donde esta especificada las sentencias, ver Función “setTimeout”.

Como se puede observar, tomará 3 segundos para que se ejecute la función llamada funcionConRetraso, y esto pasará solo una vez.

**“SetTimeout”**

1. <! DOCTYPE html>
2. <html>
3. <head>
4. </head>
5. <body>
6. <script type="text/JavaScript">
7. function funcionConRetraso(){
8. alert("Han pasado 3 segundos.");
9. }
10. setTimeout(funcionConRetraso, 3000);
11. </script>
12. </body>

</html>

* + **Bloque “setInterval”**

Es una función que está determinada para repetir una funcionalidad en un rango de tiempo definida, esta función recibe 2 parámetros, uno de ellos es el tiempo de ejecución y debe estar expresado en milisegundos, el otro es una función, donde esta especificada las sentencias, ver figura Función “setInterval”.

Como se puede observar, cada 3 segundos se ejecutará la función llamada funcionQueSeRepite, y esto pasará x cantidad de veces hasta que no se detenga el proceso del intervalo.

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

function funcionQueSeRepite(){

alert("Han pasado 3 segundos.");

}

setInterval(funcionQueSeRepite, 3000);

</script>

</body>

</html>

Estas funciones definidas por el “core” de JavaScript son sumamente útiles, de tal forma que el **“setTimeout”** se usa generalmente para establecer configuraciones, que se deben ejecutar después de cierto tiempo, es decir, si se renderiza en el navegador el html, es pertinente esperar algunos milisegundos para que el DOM pueda identificar todos los objetos creados, ahí se ejecutan las funciones de asignación de listeners entre otros, mientras que los “**setInterval”,** se usan para simular procesos en tiempo real, es decir consultar concurrentemente al servidor algún cambio de estado o también para actualizar la información de algún componente en el “front”.

## Nombramiento y funcionamiento

Algunos lenguajes son multiparadigma, es decir que el estilo de programación es funcional con diferentes paradigmas de programación, entre los más comunes es el procedimental y el orientado a objetos, de esta manera se presenta:

1. Al ser un lenguaje de programación no tipado y al mismo tiempo ser tipo script, permite implementar el paradigma procedimental o de la misma manera el orientado a objetos.

El decir que es mejor el desarrollo procedimental es correcto, pero decir que es mejor el orientado a objetos también.

1. El paradigma procedimental, hace referencia a la definición de bloques de código globales que pueden ser ejecutados en cualquier parte del desarrollo, es decir la construcción de funciones y procedimientos genéricos que se pueden reutilizar tomando como base el negocio y los dominios del problema que se pretende solucionar.
2. Acá solo será necesario definir funciones y/o variables generales que permitan realizar control de estados globales y procesos genéricos.
3. El paradigma de programación orientado a objetos conlleva al uso de estructuras más especializadas para realizar el desarrollo, de la misma forma que el procedimental, estas estructuras son globales y se pueden reutilizar en toda la aplicación.
4. En POO se utilizan clases, que a su vez permite implementar herencia, polimorfismos, sobrecarga, sobre escritura, atributos, métodos, constructores e instancias de objetos, entre otros conceptos que no existen en el paradigma procedimental.
5. El uso de todas estas cualidades hace que un proyecto de “software” sea ordenado y escalable y a su vez robusto.

Expuesto todo lo anterior, se puede llegar a concluir que POO es un concepto más avanzado para la construcción de aplicaciones, y eso es totalmente correcto, el tema es que muchas de las soluciones en JavaScript son más sencillas que al final emplear siempre POO que complicarían la misma interactividad y el desarrollo de plantillas o páginas “webs” sencillas. Ahora, si la misión es desarrollar una aplicación web, será necesario utilizar POO, todo dependerá del enfoque que se requiera para el proyecto.

## Ciclos

Hasta el momento se han empleado estructuras secuenciales y condicionales, pero, existen otro tipo de estructuras tales como son los ciclos, o las estructuras repetitivas. Estas estructuras permiten ejecutar repetidamente una cantidad de veces necesarias de un bloque de código sin necesidad de escribir de nuevo su código.

Este tipo de estructuras se caracteriza por tener las siguientes características:

* Condición para ingreso al bloque de código.
* Cambio de variable de control para salir del ciclo.
* Sentencias para ejecutarse por cada repetición.

Observa aspectos que se deben considerar en los ciclos.

**[Enlace del video](https://www.youtube.com/embed/YRPvdpLNH5g?si=7LGe07Eat1w9pP5a)**

**Ciclo” while”**

Esta estructura presenta una condición similar al del condicional if, el cual evalúa las condiciones iniciales para dejar ingresar al ciclo, dado que la condición sea verdadera se ingresara al ciclo, en el interior de la estructura se debe garantizar que se realice el cambio de valor de la variable de control del condicional, dado que si este valor nunca cambia y la evaluación de la condición es igual al requerido para ingresar esto generará un ciclo infinito, o como también es conocido un bucle infinito, este tipo de estados es crítico, dado que al dejar un ciclo infinito que consume x cantidad de recursos permitirá que la memoria del programa o en el peor de los casos de la máquina colapsen y lleguen las famosas pantallas azules.

1. “While” infinito

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

var suma = 0;

var contador = 0;

while(contador < 10) {

var num = prompt("Ingrese número: ", "");

suma += num;

}

window.write("La suma de los números ingresados es: " + suma);

</script>

</body>

</html>

En la figura 16 se puede observar la palabra reservada “while”, seguido de paréntesis y una expresión relacional, la cual pregunta si la variable contadora es menor que 10. Cuando se ejecuta la lógica el valor de la variable es 0 por lo cual esa sentencia devolverá el valor de verdadero, por lo cual entrará al cuerpo del ciclo, en las instrucciones del ciclo se captura un número cada vez que pasa por una iteración y posteriormente se realiza una sumatoria, esta sumatoria se imprimirá apenas se salga del ciclo. Es importante recordar que cuando el flujo llegue la llave que cierra el ciclo él se devolverá a la parte superior del ciclo y volverá a preguntar por la sentencia establecida, en este caso como en el cuerpo del ciclo la variable contadora no cambio, es decir su valor sigue siendo 0, el resultado de esa pregunta será verdadero ingresando nuevamente al ciclo, así de esta manera de forma infinita.

1. “While” con iteraciones definidas

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

var suma = 0;

var contador = 0;

while(contador < 10) {

var num = prompt("Ingrese número: ", "");

suma += num;

contador ++;

}

window.write("La suma de los números ingresados es: " + suma);

</script>

</body>

</html>

En la figura 17 se realiza un cambio en la lógica del código expuesto, el cual consiste en realizar un cambio de valor de variable, de la variable de control de la sentencia condicional del ciclo, por lo cual, el valor de esta variable se modificará de tal manera que cada vez que pase por el ciclo se le añada una unidad, es decir que cuando salga de la primera iteración su valor no será 0, si no 1, y para la siguiente iteración será 2, y así sucesivamente, por lo que en algún momento su valor llegara a 10, en ese caso al realizar la evaluación de preguntar si 10 es menor que 10, el resultado de la sentencia será falso y se saldrá del ciclo, al salir del ciclo la sentencia que está por fuera, imprimirá el acumulado de los números ingresados por el usuario.

**Ciclo “do-while”**

1. Ciclo “do-while”

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

var suma = 0;

do {

var num = prompt("Ingrese número: ", "");

suma += num;

var opcion = prompt("Continuar? si/no: ", "");

} while (opcion === "si");

window.write("La suma de los números ingresados es: " + suma);

</script>

</body>

</html>

**Ciclo “for”**

La estructura **“**for” es una estructura más sencilla que regularmente es usada para ejecutar problemas relacionados a ciclos iterativos numéricos, aunque el “while” y el “do-while” también lo pueden realizar, esta estructura tiene los elementos incorporados para realizar el proceso más sencillo. La estructura del **“**for” contiene en tres subestructuras o argumentos, las cuales son el iniciador de la variable de control, la condición de la variable de control y el incremento/decremento o modificación de la variable de control; su uso es aplicado cuando se conoce el valor de las iteraciones ya sea porque se obtiene de una constante, de una variable o de un valor literal, ver figura 19.

1. Ciclo **“**for”, con 10 iteraciones fijas

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

for(var i=0; i<10; i++) {

var num = prompt("Ingrese número: ", "");

suma += num;

}

window.write("La suma de los números ingresados es: " + suma);

</script>

</body>

</html>

Es de recordar que hasta el momento se han determinado diferentes estructuras de control, y es posible que, dependiendo de la necesidad a solucionar, sea necesario que una anide a otra, por decirlo, un condicional anide un ciclo, un ciclo anide a un condicional, un ciclo anide a otro ciclo, etc.

## Bloques de código

Los bloques de código son una forma de organizar el código fuente de una aplicación, en JavaScript después de varias décadas de haber salido a la luz, todavía no es muy clara la forma de definir los bloques, aunque a nivel general los desarrolladores después de proyecto insignia, han definido algunos estándares que se siguen a menudo por la comunidad.

Los bloques de código se determinan por las llaves ‘{’ y ‘}’, estos están presentes en declaración literal de objetos, condicionales, ciclos, funciones, entre otros, se podría pensar que estos solo se pueden abrir en ese tipo de estructuras, pero la verdad es que esto no es del todo cierto, ya que un bloque de código se puede abrir en cualquier parte del código.

Esto permite realizar la organización de código, acorde a las necesidades del equipo o del desarrollador como tal (Ver figura 20).

1. Bloques de código

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

function test(param1, param2) {

declareVars: {

var myConstant = 100,

result;

}

processing: {

result = myConstant + param1 + param2;

}

printResult: {

return result;

}

}

Let result = test(10,20); //130

console.log(result)

</script>

</body>

</html>

Como se puede observar en la anterior figura, se definen 3 bloques de código, los cuales no son necesarios, ni tampoco interfieren con la lógica funcional de la función test, que se usa para organizar las zonas internas de la función. En JavaScript, hay más de una forma en hacer cada cosa, por ello es importante tener en cuenta cada una y definir cuál de todas será la que se usará en el desarrollo de los proyectos que llevará a cabo.

## Prototipos

Son estructuras que permiten modelar datos que pueden heredar características entre sí similares a las clases en lenguajes altamente tipados como Java, es decir son estructuras que representan la tipología para la agrupación de información, compuesta de atributos, constructores y métodos. Tal como una función, se puede declarar un prototipo, el cual para posteriormente permitirá utilizar un constructor para establecer una instancia tal como lo muestra la figura 21.

Como previamente se comentó, desde el 2015 se incorporaron las clases al lenguaje de JavaScript y con ello características propias de la POO, pero antes de esto se hacía uso de los prototipos que eran la forma como JavaScript podía heredar funcionalidades de otro objeto.

1. Prototipo JavaScript

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

punto2D.prototype.mover = function(dx, dy) {

this.x += dx;

this.y += dy;

}

var p1 = new Point2D(1,2);

p1.mover(3,4);

console.log(p1.x); // 4

console.log(p1.y); // 6

</script>

</body>

</html>

## Funciones y declaración de funciones

En la programación de “software”es común tener fragmentos de códigos, que permitan dividir o segregar la lógica. Estos elementos de códigos tener una particularidad funcional o procedimental, de ahí que se nombren procedimientos o funciones. En JavaScript los procedimientos y las funciones, presentan pocas diferencias, dado que es un lenguaje de programación no tipado.

Una función es un conjunto de instrucciones que resuelven una parte del problema y pueden ser llamados desde diferentes partes del programa, es decir, que en varias partes del programa se requiere realizar la sumatoria de 5 números, no sería ideal que por cada vez que se requiera hacer ese proceso se duplique el código de la suma, si no que por el contrario, se define una función que reciba como argumento los valores que se desean operar y devuelve el valor, de esta manera si en la lógica de negocio hay 10 partes en las que se requiere hacer la suma de los 5 números, no será necesario escribir 10 veces el código, si no que se invocará o llamará 5 veces el bloque de código definido como función, esto conlleva una serie de beneficios, tales como dividir los problemas, reutilizar el código, segregar las operaciones, desacoplar las funcionalidades, entre otros, ver figura 22.

1. Función de retorno

<! DOCTYPE html>

<html>

<head>

</head>

<body>

<script type="text/JavaScript">

function sumar(n1,n2,n3,n4,n5) {

var suma = n1 + n2 + n3 + n4 + n5;

return suma;

}

var s1 = sumar(1,1,1,1,1); // 5

var s2 = sumar(1,2,3,4,5); // 15

var s3 = sumar(9,1,2,0,-3); // 9

</script>

</body>

</html>

En la anterior figura se puede observar, que en la parte inicial del “script” se define una función de retorno denominada sumar, la cual recibe 5 parámetros presuntamente numéricos y que la función devuelve la suma de esos 5 argumentos. En la parte inferior del “script”se puede evidenciar 3 llamados a la función sumar, pasando en cada llamado valores dientes, por lo que el retorno en cada llamado será diferente.

# Tecnologías para el desarrollo Node.js

Node.Js es una tecnología de desarrollo de “software”, con la que se puede construir cualquier tipo de necesidad de ámbito general, está basada en JavaScript y es totalmente particular dado que es una solución que está sacada totalmente del ámbito del navegador. Los creadores exaltaron las cualidades del lenguaje y lo abstrajeron a otro contexto haciendo uso del motor v8.

Con Node.Js se pueden construir diversas aplicaciones, actualmente hay varios CRM desarrollados en esta tecnología, aunque se ha vuelto muy popular en el desarrollo de APIs tipo REST. Es una tecnología multiplataforma que puede correr en cualquier sistema operativo, ya sea Microsoft, Linux o Mac, para el desarrollo de una aplicación es sumamente sencillo solo es cuestión de instalar el npm y tener un IDE o un editor de código de preferencia y comenzar a codificar.

## “Frameworks” “back-end”

En el ámbito del desarrollo de “software” y especialmente en el desarrollo “backend” hay diversos lenguajes de programación y a su vez por cada uno de ellos hay diferentes “frameworks” de desarrollo. Un “framework backend” es el que se ejecuta del lado del servidor y determina una serie de paquetes, librerías y funcionalidades estándar para el desarrollo en este caso generalmente *web*.

Los “frameworks” sirven para agilizar el proceso de desarrollo, teniendo en cuenta que ya se cuenta con varios elementos preconstruidos y se pueden extender y reutilizar, permitiendo organizar de mejor manera el estilo de desarrollo y enfocando la mayoría de los esfuerzos en la lógica de negocio y no en configuración o tareas técnicas.

Los más usados son:

1. **PHP:** laravel o symphony, simples, potentes y presentan una interfaz sofisticada y elegante.
2. **Ruby:** ruby on rails.
3. **Python:** django, flask, dependiendo el enfoque será necesario usar uno o el otro.
4. **JavaScript:** node.Js implementando Express o Next.
5. **Java:** spring MVC, Spring Boot, Spark, entre otros.

Para definir qué lenguaje y “framework” usar, es ideal qué tipo de proyecto y con base a qué arquitectura se delineará, dado que no es igual una aplicación monolítica a una orientada a microservicios, aunque tampoco será igual, implementar un componente en Java donde se necesitaran más recursos de ejecución que en Node.Js que presenta un estilo más liviano para los servidores.

## Introducción al “back-end”

Es muy común dar bastante importancia a los aspectos de las aplicaciones *web* que son visibles, como lo son las imágenes o el diseño, pero también existen otros elementos muy importantes para que una aplicación funcione de forma correcta y que no son visibles a simple vista.

El “**backend”** es esa parte invisible pero esencial en todo proyecto *web*, y a la que es de extrema importancia darle toda la trascendencia del caso, porque de él depende en gran parte los procesos informáticos que realiza un sistema.

En el desarrollo *web*, el “backend”, es la parte del sistema que se encarga de todos los procesos necesarios para que la aplicación *web* funcione de forma correcta. Estos procesos o funciones no son visibles, algunas de estas acciones que controla el “backend” son la conexión con la base de datos o la comunicación con el servidor de y los procesos o algoritmos que debe realizar la aplicación. A continuación, se presenta como desarrollar un “backend” sencillo que permita interactuar con la base de datos sin prestar mucho énfasis en el desarrollo de algoritmos informáticos. Esto debido a que en los procos de pruebas es importante comprender como funcionan internamente las aplicaciones “web”.

## Node.Js

Es un entorno de ejecución multiplataforma de código abierto, fundamentalmente diseñado para el desarrollo del lado del servidor con el lenguaje JavaScript, sus características es que es asíncrono, una arquitectura orientada a eventos y basado en el motor V8 de Google. Se debe recordar que las soluciones desarrolladas en este entorno no se ejecutan en el navegador si no en el servidor “web”.

La particularidad de Node.Js es que incorpora una serie de módulos básicos para dar solución a problemas comunes para los desarrolladores, tales como componentes para conexión a bases de datos, componentes para crear servicios “web”, para manipulación de archivos, creación de imágenes dinámicas entre otros.

Una ventaja de conocer este “framework” es que habrá homogeneidad en el desarrollo, teniendo en cuenta que tanto el “front-end” como el “back-end” manejarán el mismo lenguaje de programación, y estructuras similares, a pesar de que las tecnologías sean diferentes y estén totalmente separadas.

1. Desarrollo del lado servidor con Node.Js

var net = require('net');

net.createServer(function (stream) {

stream.write('hello\r\n');

stream.on('end', function() {

stream.end('goodbye\r\n');

});

stream.pipe(stream);

}).listen(7000);

## Express

Es el “framework” más popular de Node.Js para la creación de componentes y funcionalidades del lado del servidor, este marco de trabajo presenta diferentes módulos y componentes, que facilitan el desarrollo.

Se pueden encontrar módulos para:

* Gestión de rutas.
* Renderización de templetes.
* Manejo de sesiones.
* Seguridad, entre otros.

Se caracteriza por simple pero altamente potente, con capacidad de crear aplicaciones robustas y con un alto performance, especialmente tratándose de funcionalidades asíncronas y de tiempo real.

## Ciclos de vida y beneficios

Node.Js se basa en el tiempo de ejecución del motor v8 de Chrome, el cual está diseñado en C++, periódicamente surgen actualizaciones, parches de seguridad, rendimiento y compatibilidad con la tecnología base de JavaScript, esto se realiza regularmente para todos los sistemas operativos. El uso de JavaScript en el “frontend” y “backend” promueve el uso de Json como transferencia de datos, haciendo que sea una comunicación bilateral más rápida.

La arquitectura de un subproceso permite controlar múltiples conexiones de manera simultánea de manera eficiente discrepando de otras tecnologías que en su caso crearían un nuevo hilo de ejecución, utilizando más memoria RAM en el proceso. Node al operar sobre un solo hilo, hace uso de bucles de eventos y devoluciones para operaciones del tipo E/S, delegando tareas tipo bloqueantes para que se resuelvan lo antes posible tales como las conexiones a las bases de datos.

Cuenta con el módulo de **Cluster**, el cual permite el equilibrio de carga en varios núcleos de la CPU, adoptando mejor escalabilidad.

Entre las ventajas podemos encontrar:

1. Es confiable crear aplicaciones en tiempo real con mucho tráfico.
2. Utilizar un solo lenguaje tanto en el servidor como en el cliente.
3. Aumenta la eficiencia en el proceso de desarrollo y se minimiza la discrepancia entre el desarrollador “front” y “back”.
4. El NPM está en constante crecimiento agregando módulos y funcionalidades a usar.
5. Se ejecuta más rápido que cualquier implementación en otros entornos.
6. Es ideal para el desarrollo de microservicios.

# Características y bibliotecas

Como se ha mencionado a lo largo del desarrollo de este componente, el desarrollo “web” está compuesto de diferentes tecnologías y componentes, aunque principalmente se debe destacar la tecnología que permite interactuar (“frontend”) y lo que permite que todo este centralizado (“backend”).

Cuando se habla de aplicación del lado cliente no solo se habla de la interfaz gráfica de los sitios o aplicaciones “web”, si no por el contrario, también de las aplicaciones de escritorio, aplicaciones móviles, aplicaciones de Smart tvs, entre otros elementos. Es por medio de estas aplicaciones que es posible interactuar, enviar comandos, indicar acciones, visualizar información, registrar información entre otros, es decir, es la capa más externa, orientada hacia el usuario y es aquí donde se puede evidenciar la experiencia de usuario.

El desarrollo de aplicaciones del lado del cliente específicamente soluciones nativas se pueden encontrar diversas tecnologías, para Android se tiene Android Studio, el cual es una serie de SDK’ donde el lenguaje principal es Java o Kotlin; mientras que para iOS se tiene xcode que al igual que Android estudio tiene sus librerías, teniendo como lenguaje de desarrollo Swift y previamente Objective C; cuando se vira al desarrollo de aplicaciones de escritorio, cada casa puede tener sus propias herramientas de desarrollo, tal como se ve en el mundo mobile, en el caso de Microsoft se cuenta con Visual Studio, y como lenguajes de programación C#, C++, entre otros; Mac como se mencionó previamente hace uso de xcode y Swift; haciendo un paréntesis en este punto, es de recordar que las aplicaciones desarrolladas para Windows no son compatibles con las aplicaciones de OSX, al igual que las desarrolladas para Android no son compatibles con iOS.

También se puede encontrar desarrollos multiplataforma, como es el caso de Java, el cual es un lenguaje interpretado y para cada sistema operativo, tiene su traductor, que en este caso es la JVM. Para el caso *mobile*, también existen diferentes “frameworks”, aunque estos por lo general se basan en visores “web”, es decir, el desarrollo es totalmente “web”, pero con funcionalidades nativas, tales como el caso de Xamarin, Apache cordoba, ionic2, React Native, entre otros.

Específicamente en el desarrollo “web”, el desarrollo “frontend” solo tiene una serie de tecnologías, las cuales son html, css y javascript, estas son las que son interpretadas y renderizadas por los motores de los navegadores. El html cumple la función estructural, por medio de su lista de etiquetas, permite establecer un documento electrónico; el css cumple la función relacionada al aspecto, permite ordenar, dar forma, dar color, tamaño, entre otras características, es por decirlo así la obra blanca de un sitio “web”; mientras que JavaScript otorga la capa funcional y lógica, todo lo relacionado al control del documento, la algoritmia y funcionalidad es gracias al lenguaje de programación. Se debe recordar que cada uno de esos documentos de forma independiente en el ámbito “front” no aporta valor al desarrollo “front”, es decir, solo el html no establece un sitio “web”, solo el css no establece un sitio “web” y solo el JavaScript no establece un sitio “web”, por lo menos deben existir dos de los componentes para poder crear algo y una de las tecnologías fundamentales en este caso es el html, entonces se puede decir que para crear algo, debe obligatoriamente existir la estructura html, pero podría faltar el css o el js.

## Conceptos TCP

Protocolo de control de transmisión por sus siglas en inglés, es un protocolo de red que permite que entre servidores haya un intercambio de información, permite que los datos lleguen en el mismo orden en el que fueron enviados para que no haya distorsión de la misma.

El rol de este protocolo es controlar que los paquetes de datos lleguen de forma ordenada, confiable y sin errores, también tiene el control de concurrencia, por lo que validara que las peticiones iniciales sean pequeñas y a la medida de la necesidad vayan aumentando poco a poco, tomando como base el ancho de banda de los servidores y las redes que puede soportar.

**“Frameworks” “front-end”**

Se comienza por explorar la siguiente información:

1. Video “Frameworks” del “Front-end”



[**Enlace de reproducción del video**](https://www.youtube.com/watch?v=Z-YzV72qq4o)

|  |
| --- |
| **Síntesis del video: “Frameworks” del “Front-end”** |
| Un “framework” es un tipo de herramienta o recurso, que hace que el desarrollo de cualquier aplicativo sea del lado del cliente o servidor, sea más rápido, fácil y robusto. Este tipo de herramientas incorporan un conjunto de librerías, materiales y adoptan una estructura que van a permitir que todas las aplicaciones tengan un mejor rendimiento.  En ese sentido, se lista los “frameworks” más importantes a la fecha con el fin de que se tenga un reconocimiento para usos futuros. Este listado es apoyado según la comunidad más grande de gestión de productos llamada Product Coalition.  React: está hecho por un desarrollador de Facebook, Jordan Walke. La biblioteca se utilizó por primera vez en el servicio de noticias de Facebook en 2011, luego en Instagram en 2012 y en 2013 el código se abrió para todos.  Vue.Js: más de 36.000 aplicaciones están escritas en Vue. Entre las empresas que lo eligieron como herramienta de desarrollo, se encuentran Stackoverflow y PlayStation, según lo define la comunidad de Product Coalition.  Angular: es en realidad un proyecto de Google que se utiliza principalmente para el desarrollo de aplicaciones móviles y de escritorio. Hay muchas funciones útiles.  El sitio oficial de “framework” dice: 'Concéntrese en crear sitios impresionantes en lugar de hacer que su código funcione'. |

Los “framework” son una lista de librerías, archivos, funcionalidades, complementos, entre otros elementos que permiten el diseño y desarrollo de sitios “web” rápidamente. Los “frameworks” implementan funcionalidades que son genéricas en el mundo del desarrollo de “software”, en el “frontend”, existen para la arquitectura funcional como los de JavaScript y para temas visuales tales como el de CSS.

Esta es una lista de “frameworks” del lado cliente:

| Tipo de “frameworks” | Nombre | Autor |
| --- | --- | --- |
| “Frameworks” de CSS | Bootstrap | Twitter |
| “Frameworks” de CSS | Material Design | Google |
| “Frameworks” de CSS | awsm.css | Amazon |
| “Frameworks” de CSS | tailwindcss | Tailwindcss |
| “Frameworks” de JS | Angular | Google |
| “Frameworks” de JS | React.js | Facebook |
| “Frameworks” de JS | Vue.js | Evan You |
| “Frameworks” de JS | JQuery | Jquery |

Es relevante que antes de iniciar el proceso de desarrollo de un sistema de información, analizar qué herramientas, componentes y “frameworks” serán usados para su implementación, esto es importante dado que ayuda a refinar los requisitos funcionales de tal manera que se aprovechen al máximo las capacidades de las herramientas a usar.

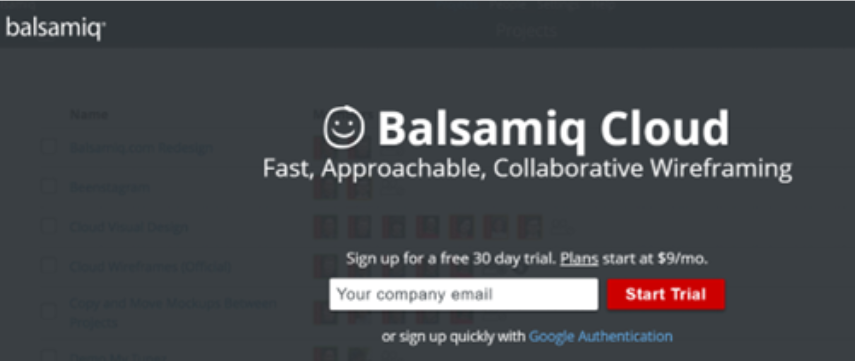
**Prototipo (Mockus, Sketch, Wireframe)**

Un prototipo es la elaboración de un producto con los componentes básicos, muchas veces solo es la representación visual de los elementos que se desarrollaran en el producto. Esta representación puede ser con técnicas de bajo nivel tales como el sketch, que son ilustraciones a mano alzada, más especializadas tipo mockups con herramientas de “software” que permiten construir vistas de forma genérica, tipo wireframe, donde toma mucha más relevancia el aspecto desde la perspectiva del diseño o funcionales con maquetas en html.

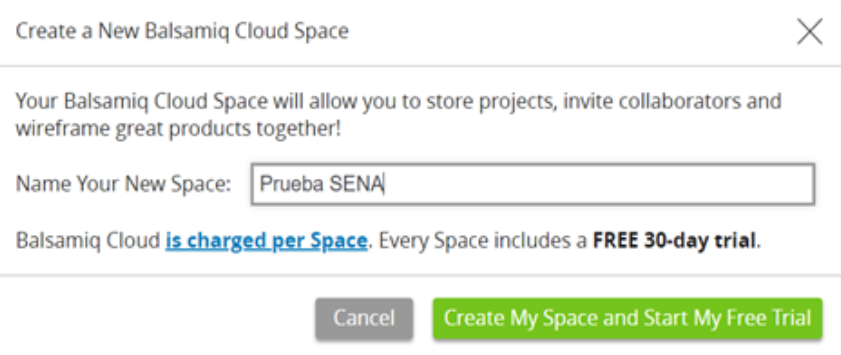
Teniendo en cuenta lo anterior, crear un proyecto tipo mockup en la siguiente herramienta:

[**Enlace del sitio web**](https://balsamiq.cloud/)

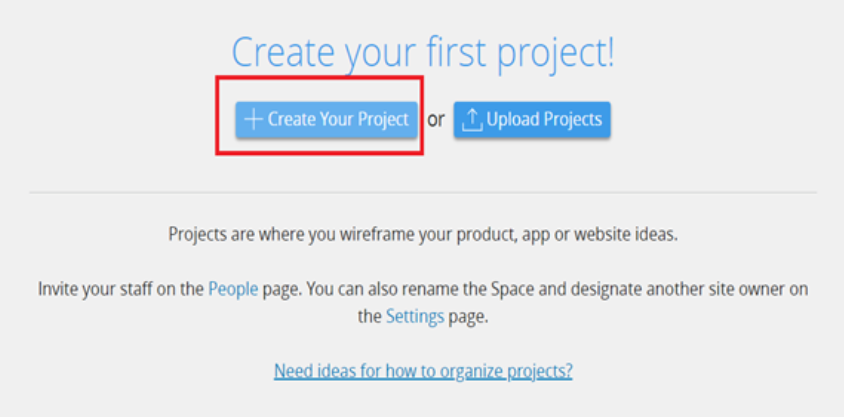
* **Proceso de registro**



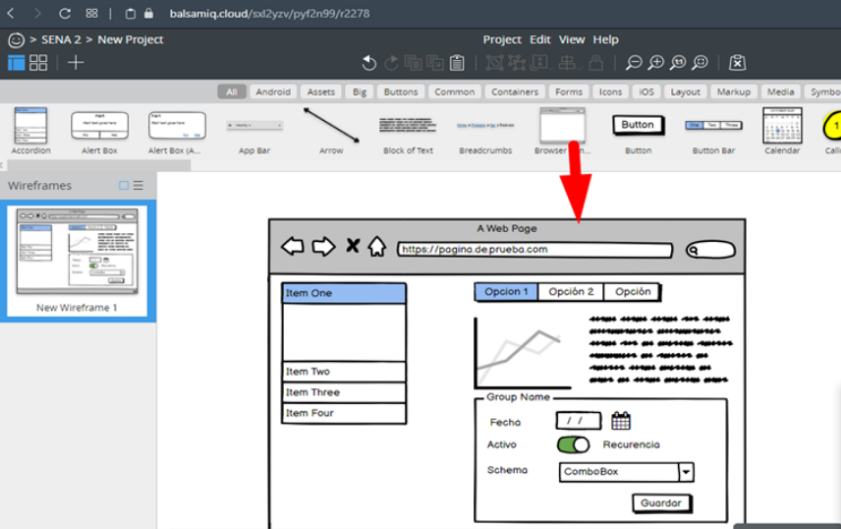
* **Creación del espacio de trabajo**



* **Creación del primer Proyecto**



* **Creación de la estructura de las vistas**

***Balsamiq***, solo es una herramienta de las muchas que se pueden encontrar en internet.

**Introducción al “front-end”**

El “front-end” hace parte del desarrollo “web”, en el cual se construye todos los componentes que están de cara al usuario, es decir todo lo visual y relacionado a la interacción de usuario, en otras palabras, lo que también se conoce como el diseño “web”. Esto tiene en cuenta el contenido, la diagramación, colores, tipografía, elementos gráficos de control, tamaños de letra, color de letra, flujos, navegación, experiencia, usabilidad, entre otros.

Para la construcción de un sitio “web”, se requieren de diferentes elementos y tecnologías, y hablando solamente del “front-end”, es requerido identificar que no se puede tener un componente sin vincular las tecnologías de html, css y javascript, las cuales en conjunto determinan el “front”.

El html determina la estructura, las bases del diseño “web”; el css otorga forma, contextura, agrega valores a las propiedades de los elementos, es decir, es la tecnología que permite dar un “look and feel” al sitio “web”; mientras que JavaScript, es la tecnología que permite dar funcionalidad al sitio; si no existiera alguno de los 3 en la construcción de un sitio “web”, sería casi imposible su óptimo desarrollo.

* **Html5, CSS3**

Es la última versión del estándar de HTML, que se caracteriza por definir las etiquetas de los documentos de la “web”, es usada para establecer la estructura fundamental de un documento “web”. Este documento compuesto de etiquetas se valida en el navegador y se renderiza para luego pintar de forma visual los elementos definidos en el código fuente.

* **CSS3**

En la última versión del estándar de CSS, la cual se caracteriza por permitir dar aspecto y forma a la estructura definida en el html, tales como tamaño, márgenes, bordes, color, fondos, entre otros, en esta versión se añaden de forma nativa animaciones y transiciones que mejoran la experiencia visual del usuario. Es una tecnología tipo script, la cual por sí sola no puede afectar a nada, debe estar acompañada o debe ir de la mano de un documento html.

**TypeScript**

TypeScript es un super conjunto de JavaScript, el cual compila en JavaScript, es decir su código fuente se escribe en Ts y luego se convierte en Js para ser interpretado. Esto se realiza automáticamente dependiente de la herramienta de pre-compilación ya sea en el “frontend” o el “backend”. Se debe recordar que el navegador “web” solo puede interpretar archivos JavaScript (ver figura 24).

Ventajas:

* Facilidad al implementar aplicaciones complejas.
* Agrega tipos, como “String”, “numbers”, “Booleans”.
* Sintaxis clara para definir clases y propiedades, permitiendo el desarrollo orientado a objetos.
* Uso de herencia e implementación de interfaces.
* Captura de errores en proceso de compilación.

1. Estructura tipo clase de TypeScript

export class CreateComponent implements OnInit {

title: string;

name: string;

lastName: string;

email: string;

password: string;

user: Userx = null;

constructor(private userService: UserxService) {}

ngOnInit(): void{}

create(): void {

this.user = {

uuid: "tttt",

title: this.title,

email: this.email,

lastName: this.lastName,

password: this.password

};

this.userService.createUser(this.user).subscribe(

(data) => {

this.user = data;

alert("Creando usuario");

},

(err) => {

console.log(err);

}

);

}

}

Este lenguaje es diseñado por Microsoft y es de distribución libre.

* **Angular2**

Es un “framework” para el desarrollo de aplicaciones “web” del lado del cliente, que está desarrollado en TypeScript, es de código abierto y el propietario es Google. Es uno de los “frameworks” más conocidos y populares en la actualidad, al estar desarrollado en TypeScript, permite hacer uso y aplicación de conceptos de programación orientada a objetos.

* **VueJs**

Es un “framework” para el desarrollo de aplicaciones “web” del lado del cliente, directamente desarrollado en JavaScript, cuenta con una arquitectura declarativa y por composición de componentes. El “framework” está centrado en la construcción de la interacción de la vista, enrutamiento, construcción, gestión de estados, entre otras.

* **ReactJs**

Es un “framework” para el desarrollo de aplicaciones “web” del lado del cliente, al igual que Angular Y VueJs, este está desarrollado directamente en JavaScript y es mantenido por Facebook, es de licencia libre, busca implementar una arquitectura MVC o MVVM.

Síntesis

El JavaScript es un lenguaje de programación muy utilizado en aplicaciones móviles, al igual que para el desarrollo de páginas “web” interactivas, pues permite añadirle múltiples funciones; este lenguaje abarca diferentes tipos de “software” lo que lo ha convertido en el lenguaje de programación más popular, por lo que se hace necesario conocer todo su funcionamiento, características, herramientas, entre otros. Por lo anterior el presente componente formativo abordo todas las temáticas necesarias para conocer a profundidad sobre este lenguaje en conjunto con otras tecnologías con la finalidad de desarrollar diversas aplicaciones necesarias para el mercado. Lo visto en el componente se esquematiza en el siguiente mapa conceptual:

Esquema gráfico que resume lo abordado en el componente formativo, indicando lo relacionado con los fundamentos de JavaScript, los cuales se compone de:

Desarrollo web.
Fundamentos: Html, Css, Js.
Paradigmas y estructuras de control: secuencial, condicionantes, funciones y bloques, programación orientada a objetos.
Frameworks: Fronted, Backend.
Despliegue.

Material complementario

| Tema | Referencia | Tipo de material | Enlace del Recurso |
| --- | --- | --- | --- |
| Tecnologías para el desarrollo Node.js | Gabriel Coding. (2020). *CÓMO INSTALAR NODE JS EN WINDOWS 10.* | Video de YouTube | <https://www.youtube.com/watch?v=BgtB31gXkoA> |
| POO y estructuras de control | Ortega, D. J. O., & Pérez, M. A. B. (2019). *Bootstrap y Laravel, herramientas para el desarrollo de aplicaciones web.* | Artículo | <http://repositorio.upsin.edu.mx/formatos/142016030030OvandoOrtegaDenzelJavier10843.pdf> |

Glosario

**“Backend”:** desarrollo que se realiza del lado del servidor, aplicaciones que centralizan principalmente la lógica del negocio, las bases de datos y los recursos de un sistema para que sean consumidos por una aplicación tipo cliente, o un “frontend”.

**“Framework”**:librería o marco de trabajo que contiene, clases, funciones, configuraciones, paquetes y componentes, que son comunes en el desarrollo de funcionalidades.

**“Frontend”:** desarrollo que se realiza del lado del cliente, aplicaciones “web”, aplicaciones de escritorio, aplicaciones móviles; estas se despliegan en la máquina del usuario. Esta es la capa o el nivel con lo que los usuarios interactúan.

**JavaScript:** lenguaje de programación tipo “scripting” que sirve para agregar funcionalidad a documentos “web”.

**“Localhost”:** en el contexto de las redes de computadoras, “localhost” es una computadora que ejecuta un programa. La computadora que se tiene al frente funciona también como un servidor virtual. Este modo es comúnmente utilizado para realizar pruebas en el desarrollo “web”.

**TypeScript:** lenguaje de programación creado por Microsoft, que da supercapas a JavaScript, se caracteriza por añadir características de lenguaje orientado a objetos.

Referencias bibliográficas

Choque, J. C. (2016). *Adición de estándares de codificación node.js*. <https://gitlab.agetic.gob.bo/dbarra/bolivia-libre/commit/1ff84b6ae70f0e9436f6e6ddc7f3e060bc9556ad>

Chrome web store. (2021). *Postman*. <https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop?hl=es-419>

Colaboradores de los proyectos Wikimedia. (2020). *Framework para aplicaciones web - Wikipedia, la enciclopedia libre.* <https://es.wikipedia.org/w/index.php?title=Framework_para_aplicaciones_web&oldid=128980356>

Express. (2021). *Direccionamiento básico de Express*. <https://expressjs.com/es/starter/basic-routing.html>

Fundéu RAE. (2012). *Edición y maquetación de documentos web*. <https://www.fundeu.es/escribireninternet/edicion-y-maquetacion-de-documentos-web>

Gomez, D. (2017). Nodemon. *vortexbird*. <https://vortexbird.com/nodemon/#:%7E:text=Nodemon%20es%20una%20utilidad%20que,desarrollo%20de%20aplicaciones%20en%20nodojs>

López, A. (2021). *Qué es Postman y primeros pasos*. *OpenWebinars.net*. <https://openwebinars.net/blog/que-es-postman/>

Material Design. (2021). *The color system*. <https://material.io/design/color/the-color-system.html#color-usage-and-palettes>

Mozilla. (2021b, junio 9). *Introducción a Express/Node - Aprende sobre desarrollo web | MDN. MDN Web Docs*. <https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs/Introduction>

Mozilla. (2021b). *await - JavaScript | MDN. MDN Web Docs*. <https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/await>

NodeJS.org. (2021). *Acerca de Node.js. Node.js*. <https://nodejs.org/es/about/>

npm. (2021). *About npm*. <https://www.npmjs.com/about>

Red Hat. (2021). *¿Qué es y para qué sirve un IDE?* <https://www.redhat.com/es/topics/middleware/what-is-ide>

Whyte, A. (2021). Google JavaScript Style Guide. *Google JavaScript Style Guide*. <https://google.github.io/styleguide/javascriptguide.xml>

Créditos

| Nombre | Cargo | Regional y Centro de Formación |
| --- | --- | --- |
| Claudia Patricia Aristizábal | Líder del Ecosistema | Dirección General |
| Liliana Victoria Morales Gualdrón | Responsable de línea de producción | Centro de Gestión De Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital |
| Gustavo Adolfo Rodríguez Quinayas | Experto Temático | Centro de Teleinformática y Producción Industrial - Regional Cauca |
| Leydy Jhuliana Jaramillo Mejía | Diseñador instruccional | Centro de Gestión Industrial - Regional Distrito Capital |
| Rafael Neftalí Lizcano Reyes | Asesor pedagógico | Centro Industrial del Diseño y la Manufactura - Regional Santander |
| Carolina Coca Salazar | Asesor pedagógico | Centro de Diseño y Metrología - Regional Distrito Capital |
| Jhon Jairo Rodríguez Pérez | Corrector de estilo | Centro de Diseño y Metrología - Regional Distrito Capital |
| Yuly Rey | Diseñador web | Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital |
| Jhon Jairo Urueta Alvarez | Desarrollador fullstack | Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital |
| Ernesto Navarro Jaimes | Animación y producción audiovisual | Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital |
| Lady Adriana Ariza Luque | Animación y producción audiovisual | Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital |
| Laura Gisselle Murcia Pardo | Animación y producción audiovisual | Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital |
| Carolina Coca Salazar | Evaluación de contenidos inclusivos y accesibles | Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital |
| Lina Marcela Pérez Manchego | Validación de recursos educativos digitales | Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital |
| Leyson Fabian Castaño Pérez | Validación de recursos educativos digitales | Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital |