



El futuro digital  
es de todos

MinTIC



# Ciclo 3: Desarrollo de Software

06  
Front-End

```
element* item = el->FirstChildElement(); item != 0; item = item->NextChildElement()
{
    boost::stringref el_name = item->Attribute( "name" );
    boost::stringref type = item->Attribute( "type" );
    ...
    std::string str;
    float x = boost::lexical_cast<float>( item->Attribute( "x" ) );
    float y = boost::lexical_cast<float>( item->Attribute( "y" ) );
    float offset = boost::lexical_cast<float>( item->Attribute( "offset" ) );
    ...
    spriteDescList::iterator sp = sprite_descs.begin();
    while( sp != sprite_descs.end() && sp->name_ != spritename )
        ++sp;
}
```



El futuro digital  
es de todos

MinTIC

# Objetivo de Aprendizaje

**Identificar** los principales elementos asociados a la construcción de un **componente de presentación** (front-end), usando los lenguajes **JavaScript, HTML y CSS**, y el framework **Vue.js**.



El futuro digital  
es de todos

MinTIC

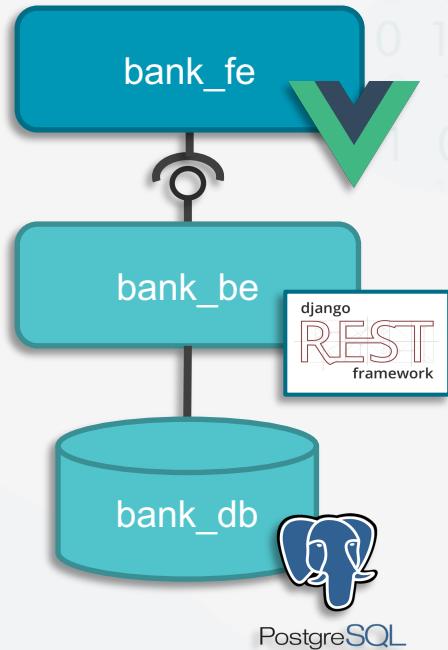
# Parte 1



# Capa de Presentación

En la **capa de presentación** se construirá un **componente** (`bank_fe`) cuya función será la de permitir la **interacción** con el **usuario final**, mediante una **interfaz gráfica** de tipo **web**.

En este caso, se hará uso de los lenguajes **HTML**, **CSS** y **JavaScript**, y del framework **Vue.js**.





# HTML: Maquetación Web

## HTML



HyperText Markup Language o HTML, es un **lenguaje de marcado** que se utiliza para **definir el contenido** y la **estructura** de una **página web**. Dicha estructura se refiere al uso de títulos, párrafos, imágenes, listas y tablas.

- HTML **puede** ser **asistido** por tecnologías como CSS y JavaScript.
- Se debe tener en cuenta que un **lenguaje de marcado** es **diferente** a un **lenguaje de programación**.



El futuro digital  
es de todos

MinTIC

# CSS: Estilos

Cascading Style Sheets o CSS, es un **lenguaje de hojas de estilo** que se utiliza para **aplicar estilos** de manera selectiva a elementos en **documentos HTML**. CSS permite la **estilización** de páginas web, y por ende, la creación de páginas con un **diseño llamativo** y elaborado.

CSS



[Imagen] CSS3 logo. (s. f.). [PNG]. Wikimedia. [https://upload.wikimedia.org/wikipedia/commons/thumb/d/d5/CSS3\\_logo\\_and\\_wordmark.svg/1200px-CSS3\\_logo\\_and\\_wordmark.svg.png](https://upload.wikimedia.org/wikipedia/commons/thumb/d/d5/CSS3_logo_and_wordmark.svg/1200px-CSS3_logo_and_wordmark.svg.png)

UNIVERSIDAD NACIONAL  
DE COLOMBIA

Misión  
TIC 2022



El futuro digital  
es de todos

MinTIC

# Parte 2



El futuro digital  
es de todos

MinTIC

## JavaScript

JavaScript es el lenguaje de programación más utilizado en el desarrollo de componentes front-end de tipo web.



Misión  
TIC 2022



# JavaScript: Definición

JavaScript es un **lenguaje de programación de alto nivel**, y al día de hoy es el **lenguaje de programación más utilizado** en la **WEB** (97% de los sitios de la web usan JavaScript para ejecutar scripts en el lado del cliente).

Las características de **JavaScript** están dadas por una **especificación** llamada **ECMAScript**, la cual se actualiza cada año.





El futuro digital  
es de todos

MinTIC

# JavaScript: Historia



JavaScript apareció en 1995 en el navegador **Netscape**, su propósito inicial era **agregar interacción** a los **sitios web** desde el lado del **cliente** y no únicamente desde el lado del servidor.

JavaScript tuvo gran éxito ya que al poder **realizar** validaciones y **cálculos** en el lado del **cliente** evitaba el **envío** de **datos** al **servidor** a través de la **red**. En los años 90 la **velocidad de internet** era muy **baja** y cualquier interacción entre el cliente y el servidor **tardaba** un **tiempo** considerable.

[https://upload.wikimedia.org/wikipedia/commons/thumb/0/08/Netscape\\_icon.svg/1200px-Netscape\\_icon.svg.png](https://upload.wikimedia.org/wikipedia/commons/thumb/0/08/Netscape_icon.svg/1200px-Netscape_icon.svg.png)





# JavaScript: Usos Actuales

Inicialmente JavaScript fue pensado como un lenguaje de programación para ser usado en navegadores y poder ejecutar funcionalidades desde el lado del cliente, sin embargo a día de hoy, JavaScript es usado para muchos otros propósitos.

Un ejemplo de lo anterior es el uso de JavaScript como lenguaje de programación del lado de servidor, lo cual es posible gracias al entorno de ejecución NodeJS.





# JavaScript: Sintaxis

```
1 <script>
2   (function () {
3     window._harvestPlatformConfig = {
4       "application": "LeCompany",
5       "permalink": "https://lecompany.com/item/%ITEM_ID%"
6     };
7     var s = document.createElement("script");
8     s.src = "https://harvestapp.com/assets/plat
9     s.async = true;
10    var ph = document.getElementsByTagName("script")[0];
11    ph.parentNode.insertBefore(s, ph);
12  })();
13 </script>
```



La **sintaxis** de **JavaScript** es muy **similar** a la sintaxis de lenguajes como **Java** o **C++**, pero **mucho** más **flexible** y **ligera**, de hecho en la mayoría de casos permite **omitar** el **punto y coma**, y a diferencia de otros lenguajes como Python, **no** existen **problemas** con la **indentación**.

A **continuación** se mostrará la **sintaxis** de algunas de las **instrucciones** más **comunes** usadas en JavaScript.



# JavaScript: Variables

Actualmente **JavaScript** maneja el concepto de **variable** y **constante**, por lo cual establece la palabra reservada **LET** para definir variables y **CONST** para definir constantes.

**Antiguamente** JavaScript **no** tenía una **distinción** clara y hacía uso de la palabra reservada **VAR** para definir constantes y variables.

```
js index.js > ...
1 // Uso de Let y Const
2 let miVariable      = 0;
3 const miConstante  = 0;
4
5 // Uso de Var
6 var variableAntigua = 0;
7 var constanteAntigua = 0;
8
```



# JavaScript: Tipos de Datos

JavaScript maneja algunos **tipos de datos** como **String**, **Number** y **Boolean**, estos no son definidos de manera explícita, ya que JavaScript es capaz de **determinar** el tipo de dato de **manera dinámica**.

Existen otros tipos de dato como **Null** que representa un valor nulo, y **Undefined** que indica que no se tiene valor alguno.

Para determinar el tipo de dato se puede hacer uso de la instrucción **typeof**.

```
js index.js > ...
1 // Number
2 let numero = 10;
3
4 //String
5 let cadena = "hola";
6
7 //Boolean
8 let booleanoTrue = true;
9 let booleanoFalse = false;
10
11 //Null
12 let nulo = null;
13
14 //Undefined
15 let sinValor;
16
```



# JavaScript: Operadores

En **JavaScript** se tienen dos grupos principales de operadores, los **operadores lógicos** que incluyen la **negación**, la **conjunción** (AND) y la **disyunción** (OR), y los **operadores aritméticos** que incluyen la **suma**, la **resta**, la **multiplicación**, la **exponenciación**, la **división**, el **modulo**, el **incremento** y el **decremento**.

La mayoría de operadores son **binarios**, es decir necesitan dos valores para efectuar la operación, salvo la **negación**, el **incremento** y el **decremento** que son **unitarios**.

```
JS index.js > ...
1 //OPERADORES LOGICOS
2 let negacion      = !true;
3 let conjucion    = true && false;
4 let disyuncion   = true || false;
5
6 //OPERADORES ARIMETICOS
7 let suma          = 12 + 24;
8 let resta         = 12 - 24;
9 let multiplicacion = 12 * 24;
10 let exponenciacion = 12 ** 24;
11 let division     = 12 / 24;
12 let numero       = 12;
13 let incremento   = numero++;
14 let decremento   = numero--;
15
```



# JavaScript: Operadores

También se tiene **otro grupo de operadores**, llamados **operadores de comparación**, estos son **útiles** a la hora de trabajar **con condicionales y ciclos**.

Dentro de estos operadores se tiene el igual (`==`), el diferente (`!=`), el mayor que (`>`), el mayor o igual que (`>=`), el menor que (`<`), el menor o igual que (`<=`) y el estrictamente igual (`===`) que no solo compara el valor si no también el tipo de dato.

```
js index.js > ...
1 //OPERADORES DE COMPARACION
2 let igual = (12 == 24);
3 let diferente = (12 != 24);
4 let mayor_que = (12 > 24);
5 let mayor_igual_que = (12 >= 24);
6 let menor_que = (12 < 24);
7 let menor_igual_que = (12 <= 24);
8 let estrictamente_igual = (12 === 24);
9
```



# JavaScript: Arrays

A diferencia de otros lenguajes **JavaScript** permite **definir arrays** (o **arreglos**) con **distintos tipos de datos**. Esto se debe a que JavaScript maneja un tipado dinámico.

Otra característica interesante de los **arrays** en **JavaScript** es que **pueden** tener un **tamaño dinámico**, es decir que su tamaño puede variar durante la ejecución del programa.

```
js index.js > ...
1 //Definiendo un ARRAY
2 let carros = ["Saab", "Volvo", "BMW"];
3
4 //Seleccionando elementos
5 let x = carros[0];      // x = "Saab"
6
7 //Modificando elementos
8 carros[0] = "Opel";
9
10 //Agregar Elemento al Final
11 carros.push("Corsa");
12
13 //Eliminar Ultimo Elemento
14 carros.pop();
15
```



# JavaScript: Objetos

En **muchos lenguajes** se tiene el **concepto** de **diccionario**, el cual es una **colección** de parejas de **claves y valores**, dentro de **JavaScript** se tiene este mismo concepto pero es conocido como **Objeto**.

En **JavaScript** los **objetos** suelen **utilizarse bastante**, ya que permiten **organizar** distintos **datos** en una **estructura** clara y concisa.

```
js index.js > ...
1 //Definiendo un Objeto
2 const persona = {
3   nombre: "John",
4   apellido: "Ramirez",
5   edad: 50,
6   colorOjos: "azul"
7 };
8
9 //Accediendo a los valores #1
10 let nombre = persona.nombre;
11
12 //Accediendo a los valores #2
13 let apellido = persona["apellido"];
14
```



# JavaScript: Condicionales

JavaScript maneja una estructura **clásica** de **condicionales**, la cual incluye un **IF**, un **ELSE** y un **ELSE IF**. Su **funcionamiento** se bastante **similar** a la mayoría de los **lenguajes**.

Al igual que en otros lenguajes JavaScript ofrece la estructura **SWITCH**, la cual suele ser **útil** cuando se requiere una **gran** cantidad de **comparaciones** sobre un mismo **dato**.

```
JS index.js > ...
1 // Uso de IF, ELSE IF y ELSE
2 let edad = 30;
3 let resultado;
4
5 if (edad > 60) {
6     resultado = "Anciano";
7 }
8
9 else if (edad > 18) {
10    resultado = "Adulto";
11 }
12
13 else {
14     resultado = "Niño";
15 }
16
```



# JavaScript: Ciclos

JavaScript ofrece tres tipos de **ciclos**, **FOR**, **WHILE** y **DO WHILE**, estos tres **funcionan** de manera **similar** a la **mayoría** de **lenguajes**.

Pero además de los ciclos tradicionales JavaScript ofrece algunas **variantes** con las instrucciones **IN** y **OF** para los ciclos **FOR**, estas variantes **facilitan** el proceso de **recorrer** los **elementos** de un **array** o de un **objeto**.

```
js index.js > ...
1  for (let i = 0; i < 10; i++) {
2    |  // CODIGO
3  }
4
5
6  while (condition) {
7    |  // CODIGO
8  }
9
10
11 do {
12   |  // CODIGO
13 }
14 while (condition);
15
```



# JavaScript: Funciones

A diferencia de otros la lenguajes, la definición de **funciones en JavaScript** es muy **flexible**:

- En primer lugar, al **no** existir de **tipado** de datos, al momento de definir los **parámetros** de la función solamente bastará con definir sus **nombres**.
- En segundo lugar, el **retorno** de **valores** por parte de una **función no es obligatorio**, cuando se deseé **retornar un valor** simplemente se **usa** la instrucción **RETURN**.

```
js index.js > ...
1 // Definicion
2 function miFuncion(a, b) {
3 | return a * b;
4 }
5
6 // Uso
7 let x = miFuncion(4, 3);
8
```



# JavaScript: Funciones Flecha

Además de la manera **tradicional** para **definir funciones**, **JavaScript** ofrece **otra manera** de definirlas, las funciones definidas de esta manera son conocidas como **funciones flecha**.

Las **diferencias** entre estos dos tipos de funciones son un poco **avanzadas**, pero en la **práctica** para **casos sencillos** de uso, **no existen diferencias** notorias.

```
js index.js > ...
1 // Definicion Tradicional
2 function hello () {
3     return "Hello World!";
4 }
5
6 // Definicion Funcion Flecha
7 const hello = () => {
8     return "Hello World!";
9 }
10
```



El futuro digital  
es de todos

MinTIC

## JavaScript: DOM

El DOM permite que JavaScript pueda manipular el contenido de documentos HTML.



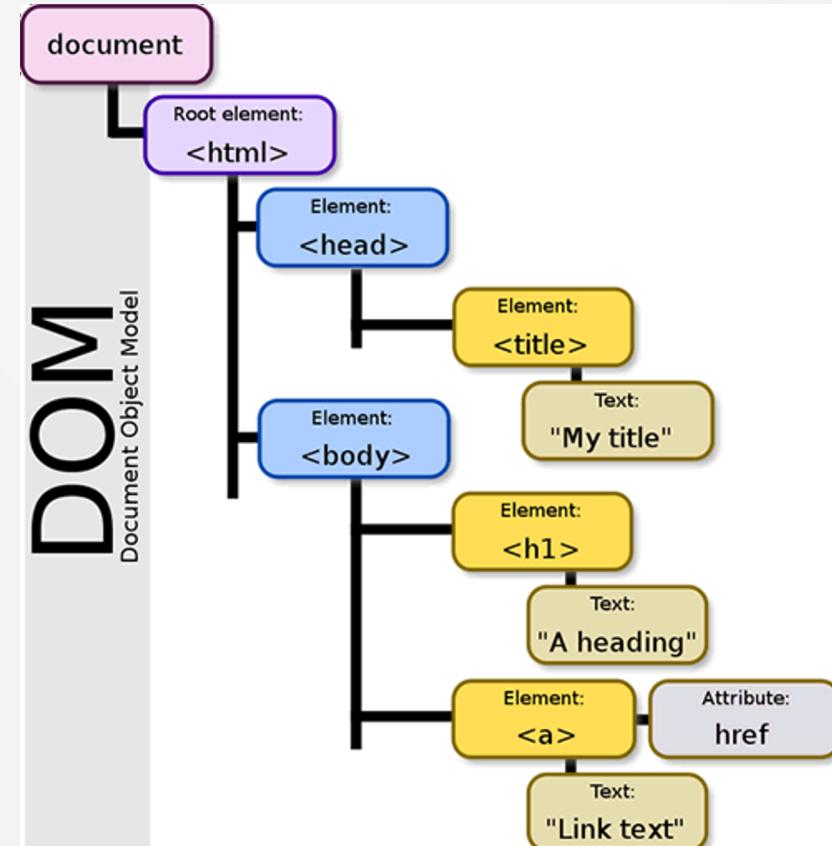
Mision  
TIC2022



# JavaScript: DOM

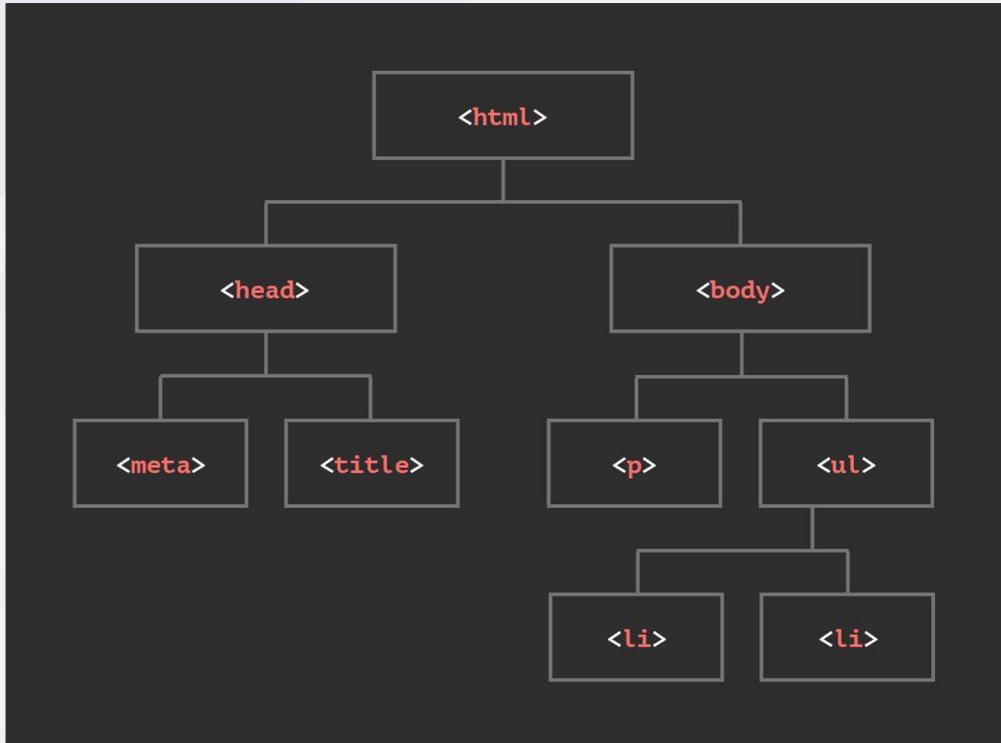
El **objetivo** de **JavaScript** en el cliente (navegador) es brindar **interactividad** a los sitios **WEB**, para ello **JavaScript** debe tener la capacidad de **manipular** y **editar** el **contenido** de los documentos **HTML** que definen la **estructura** de los sitios **WEB**.

Para llevar a cabo los **cambios** en el contenido de los sitios **WEB**, **JavaScript** hace uso del **DOM (Document Object Model)**, este **documento** define un **objeto** que **representa** la estructura del sitio **WEB**.





# JavaScript: Document



Dentro de **JavaScript** se tiene de manera **predeterminada** un objeto conocido como **Document**, este objeto **recolecta** toda la información del **DOM**, por lo cual a través de este objeto se puede **acceder** a **cualquier elemento** del sitio WEB.

Además de **recolectar** toda la información del **DOM**, el objeto **Document** brinda una serie de **funcionalidades** que facilitan la **interacción** entre **JavaScript** y **HTML**.



# JavaScript: Eventos

Una de las **interacciones** más importantes entre **JavaScript** y **HTML** es el manejo de **eventos**, un **evento** es un suceso **emitido** por **HTML** producto de la **interacción** del **usuario** con el **sitio WEB**.

Por si mismo **HTML no** es capaz de **controlar** estos **eventos**, sin embargo **JavaScript** es capaz de **capturar** dichos **eventos** y **ejecutar** ciertas **funcionalidades** cuando este suceda.

