ULAGOS

UNIDAD 2

Fundamentos de JS

INGENIERÍA CIVIL EN INFORMÁTICA
DEPARTAMENTO DE CIENCIAS DE LA INGENIERÍA

Docente: Victor Saldivia Vera Correo: victor.saldivia@ulagos.cl

ULAGOS

TABLA DE CONTENIDO

Brendan Eich creador de JS

UN POCO DE HISTORIA

JavaScript fue creado por **Brendan Eich** en **1995** mientras trabajaba en **Netscape Communications**.

JavaScript es un lenguaje de programación de alto nivel y orientado a objetos que se ha convertido en uno de los pilares fundamentales de la web. A lo largo de su historia, ha experimentado importantes hitos que han contribuido a su popularidad y evolución.

Victor Saldivia Vera | victor.saldivia@ulagos.cl

En **1997** JavaScript fue estandarizado por primera vez bajo el nombre de ECMAScript por Ecma International. ECMAScript define la especificación del lenguaje JavaScript y garantiza su compatibilidad entre diferentes implementaciones.

Luego en el año **2005**, jQuery una biblioteca de JavaScript simplificó en gran medida la manipulación del DOM (Document Object Model) y el manejo de eventos en los navegadores. jQuery se convirtió en una de las bibliotecas más populares y facilitó el desarrollo web en múltiples navegadores.

EXPANSIÓN HACIA EL LADO DEL SERVIDOR

Con el surgimiento de **Node.js** el año 2009 como entorno de tiempo de ejecución basado en el motor de JavaScript V8, significó la expansión de JavaScript, lo que permitió ejecutar el lenguaje en el lado del servidor. Con el nacimiento de Node.js brindó a los desarrolladores la capacidad de crear aplicaciones web de alto rendimiento y escalables en JavaScript.

2009

2023

Con el precedente de Node.js, surgieron más frameworks de JavaScript se han convertido en referentes en el desarrollo de aplicaciones web modernas. Angular, desarrollado por Google, React, creado por Facebook, y Vue.js, un framework progresivo, han simplificado la creación de interfaces de usuario interactivas y reactivas. Sin nombrar a muchos más entornos que existen en la actualidad.

ES6 and JS
ECMAScript 6 Javascript

Una hito importante en la historia de JavaScript es la nueva versión de **ES6**. Este estándar indica como debe ser interpretado el lenguaje, en cada tecnología como: navegadores, servidores de node.js, en el desarrollo de aplicaciones, etc

El salto de **ES5** a **ES6** significó muchas mejoras del lenguaje por ejemplo: simplificación en el proceso de código orientado a objetos, el uso de las arrows functions, el uso de promesas, y las nuevas declaraciones de variables: let y const. Actualmente ya nos encontramos en la versión ES14.

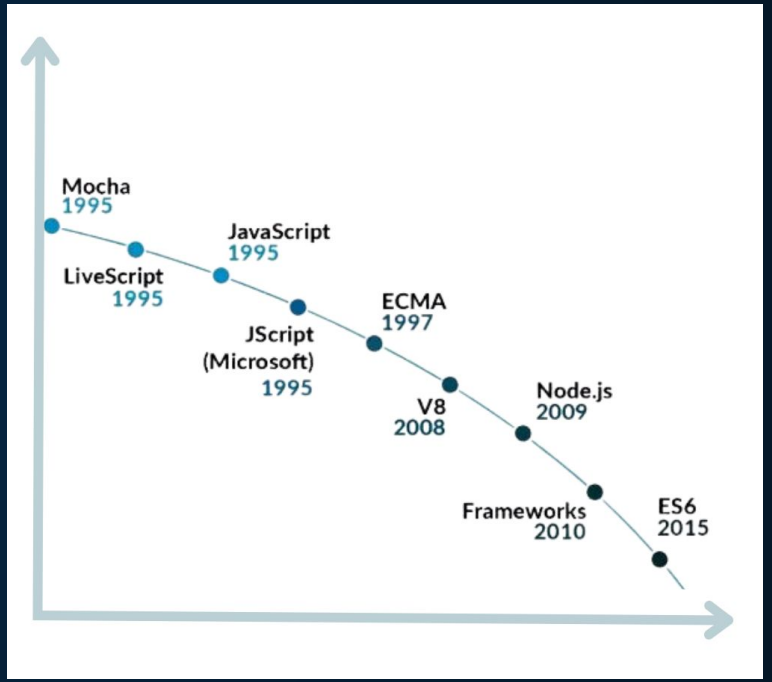
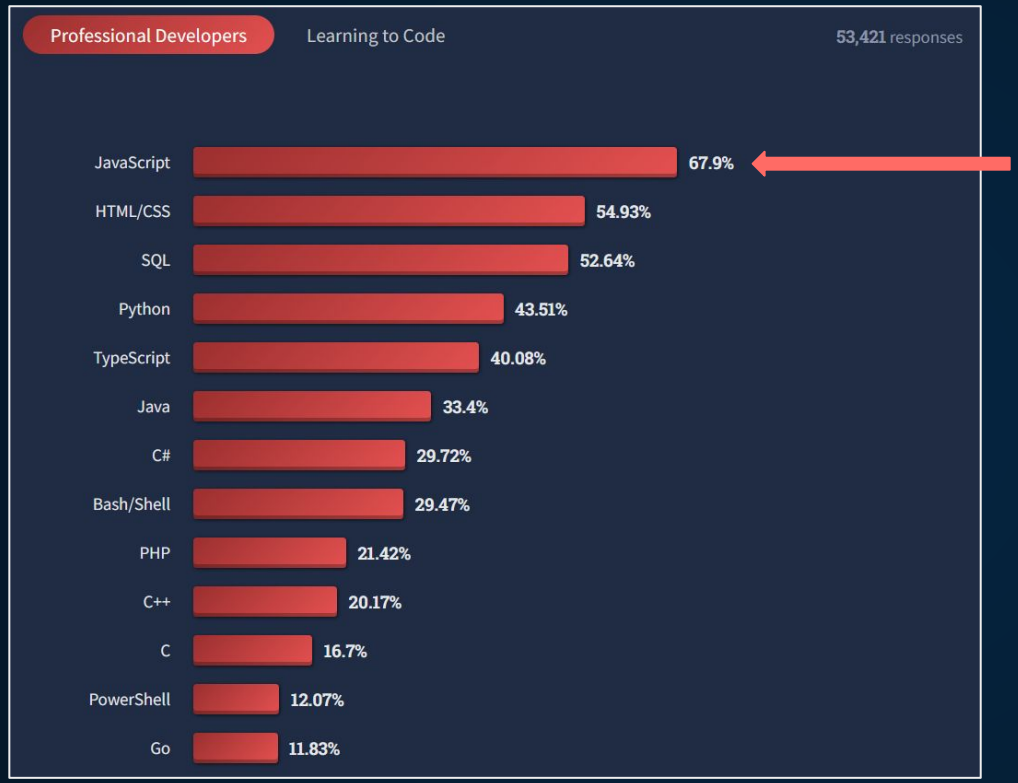


Figura 1. Línea de Tiempo de hitos más relevantes de JS

Victor Saldivia Vera | victor.saldivia@ulagos.cl

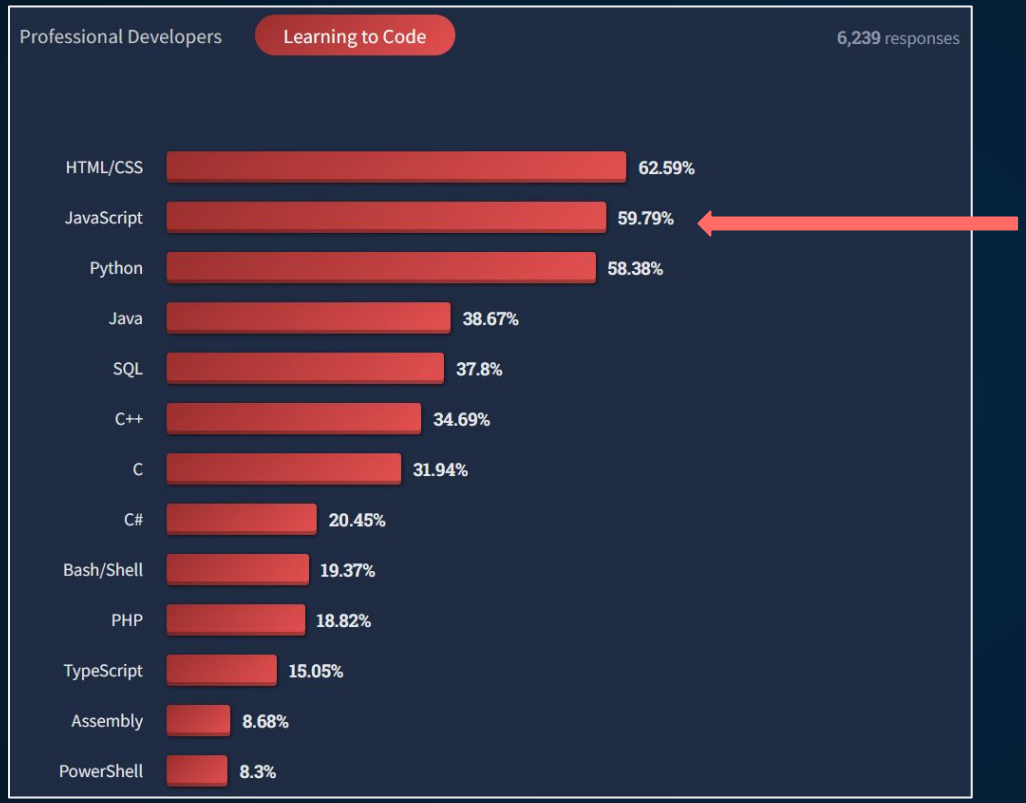


Fuente: StackOverflow - Lenguajes Más Populares por Developers 2022

El mercado actual nos indica que es indispensable para cualquier profesional del área de desarrollo conozca **JavaScript**.

El 2022 se **consagró como el décimo año consecutivo** en el que **JavaScript** ocupó el lugar del lenguaje de programación más utilizado. Es el más popular del mundo, con más de 14 millones de profesionales trabajando con este lenguaje en todo el planeta. Además, es el lenguaje de los navegadores web.

Según la encuesta realizada por **StackOverflow** el **67,9%** de profesionales tech trabajan con JavaScript.



Fuente. StackOverflow - Lenguajes Más Populares por Principiantes 2022

Importante destacar que **JavaScript** es uno de los lenguajes de programación con una curva de aprendizaje baja, por eso es uno más recomendado para empezar, porque es muy versátil y amigable con las personas que no tienen ninguna experiencia a la hora de programar.

Según la encuesta de **StackOverflow** con un **59,79%** de los encuestados principiantes, prefieren a JavaScript como lenguaje de programación a la hora de dar sus primeros pasos en el área de la programación.

Victor Saldivia Vera | victor.saldivia@ulagos.cl

TÉRMINOS A ENTENDER

ANTES DE EMPEZAR CON VARIABLES DE JS

SCOPE (ALCANCE)

El scope en JavaScript se refiere al alcance o contexto en el cual una variable o función es accesible. Determina la visibilidad y disponibilidad de las variables y funciones en diferentes partes de un programa. En **JS,** existen tres tipos principales de scope: **scope global**, **scope de bloque** y **scope local**, las cuales se explicarán en capítulos, más adelante.

HOISTING

Es un comportamiento de JavaScript, que se refiere a la forma en que el lenguaje mueve las declaraciones de variables y funciones al principio del ámbito de ejecución, antes de que se ejecute el código. Esto quiere decir, que se puede usar una variable o una función antes de haber sido declarada, aunque no es una buena práctica.

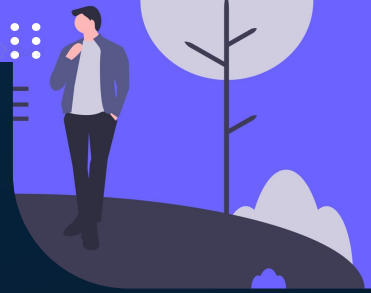
Determina el **conjunto** de variables que **podemos acceder**

JS tiene **Scope Léxico**
(Lexical Scoping)

Scope de JavaScript

JS

El scope de cada variable se determina **leyendo** el **código** del programa, **sin ejecutarlo**



Cuando hablamos de **Scope** nos referimos al **entorno**

En cada **función** héroe significa una algo **diferente**

```
function teamCap () {
    var heroe = "Capitan America";
    console.log(heroe);
    // Capitan America
}

function teamIronMan () {
    var heroe = "Iron Man";
    console.log(heroe);
    // Iron Man
}
```

Scope de JavaScript

JS

En JS tenemos Diferentes Scopes

Global Scope

Local Scope

- Block Scope
- Function Scope

Victor Saldivia Vera | victor.saldivia@ulagos.cl

Scope de JavaScript

JS

Global Scope

Cuando **declaramos fuera** de toda **función** o **bloque de código**

```
var comic = 'X-Men'

function comprar() {
  console.log(`compraste el comic ${comic}`)
  // compraste el comic X-Men
}

function leer() {
  console.log(`estas leyendo ${comic}`)
  // estas leyendo X-Men
}

comprar()
leer()
```

La variable comic se **podrá** usar a lo largo de **todo** nuestro **programa**

Block Scope

Bloque de código como ser un if, while, for, etc. es decir que están **entre llaves**

```
function mutantes() {
  const mutante = 'Wolverine'
  if (true) {
    // esta variable se quedara solo
    // entre estas llaves
    const mutante = 'Ciclope'
    console.log(mutante)
    // "Ciclope"
  }
  console.log(mutante)
  // "Wolverine"
}
```

variables **declaradas** con **let** o **const**

Victor Saldivia Vera | victor.saldivia@ulagos.cl

Function Scope

Accesibles dentro de toda la **función,** pero **no fuera** de la misma

```
function justiceLeague() {
  var heroes = ['Batman', 'Superman', 'Aquaman']
  console.log(heroes)
  // ["Batman", "Superman", "Aquaman"]
}

function teenTitans() {
  var heroes = ['Robin', 'Raven', 'Terra']
  console.log(heroes)
  // ["Robin", "Raven", "Terra"]
}

console.log(heroes)
// ReferenceError:
// heroes is not defined
```

variables **declaradas** con **var**

Scope de JavaScript

JS

Para evitar que las funciones se vuelvan **impredecibles** debemos **declarar dentro del scope más reducido**

JS

ULAGOS

01. VARIABLES

VARIABLES

En JavaScript, existen 3 formas de declarar variables: **var**, **let** y **const**. Cada una tiene sus propias características y diferencias que se explica a continuación:

VAR

Permite la reasignación de valores y tiene un scope global o de función. Es sometida a hoisting.

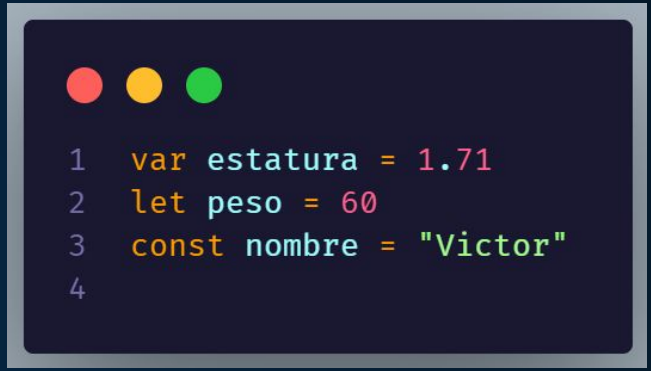
LET

Permite la reasignación de valores y tiene un scope de bloque. No está sometida a hoisting.

CONST

No permite la reasignación de valores y puede tener un scope global, de función o de bloque. No es sometida a hoisting

Antes de la llegada de **ES6 (ECMAScript 6)**, las declaraciones **var** eran las que se utilizaban. Sin embargo, hay problemas asociados relacionados con variables declaradas con var. Por eso fue necesario que surgieran nuevas formas de declarar variables, y es así donde aparece **let** y **const**.



```
1  var estatura = 1.71
2  let peso = 60
3  const nombre = "Victor"
4
```

Código 1. Variables en JS

VAR v/s LET

Uno de los problemas es, cuando se declara una **variable** con **var** dentro de una función o fuera de cualquier bloque, su alcance es la función o el ámbito global. Esto puede causar problemas de colisión de variables y dificultar la mantención del código. Las variables declaradas con **var** se mueven al principio de su ámbito **(hoisting),** esto puede generar comportamientos inesperados y dificultar la comprensión del código.

Por último, las variables **var** pueden ser reasignadas y redeclaradas dentro del mismo ámbito, lo que puede llevar a errores sutiles. Aquí es donde aparece la variable **let,** solucionando estos problemas y ofreciendo un comportamiento más predecible y seguro en cuanto a scope y hoisting.

En scripts más antiguos, siempre van a encontrar **var** en lugar de **let**:

Código 2. Variable de tipo var

En Resumen...

Es recomendable utilizar **let** en lugar de **var** en la mayoría de los casos, porque entrega un mejor control sobre el ámbito de las variables y evita problemas futuros en el código. Sin embargo, en contextos más antiguos o específicos, donde se requiere el comportamiento de **var**, aún se puede utilizar sin problemas.

¿CÓMO IMPRIMIR UN MENSAJE EN JS?

alert ( ) 

console.log ( ) 

document.write ( ) 

LA ETIQUETA SCRIPT

Los programas de JavaScript se pueden insertar en casi cualquier parte de un **documento HTML** con el uso de la etiqueta **<script>**, como se muestra el el **Código 3**.

La etiqueta **<script>** contiene código JavaScript que se ejecuta automáticamente cuando el navegador procesa la etiqueta.

Más adelante se explicará en detalle la estructura de una página web en HTML, solo basta con saber la estructura básica y fundamental para ejecutar un código JS.

```
1  <!DOCTYPE HTML>
2  <html>
3      <body>
4          <p>Antes del script ... </p>
5              <script>
6                  alert('¡Hola, mundo!');
7              </script>
8          <p> ... Desp és del script .</p>
9      </body>
10 </html>
```

Código 3. Hola Mundo en JS

SCRIPT EXTERNO

Como regla general, solo los **scripts más simples** se colocan en el **HTML**. Los más complejos están en archivos separados. La ventaja de un archivo separado es que el navegador lo descargará y lo almacenará en caché.

Si tenemos mucho código de JavaScript, es de buena práctica colocarlo en un archivo aparte.

Los archivos de **script** se adjuntan a **HTML** con el atributo **src**, como se muestra en el **Código 4**.

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>
    <!-- Hola Mundo Externo desde un Archivo JS -->
    <script src="../JS/01.holamundo.js"></script>
</body>
</html>
```

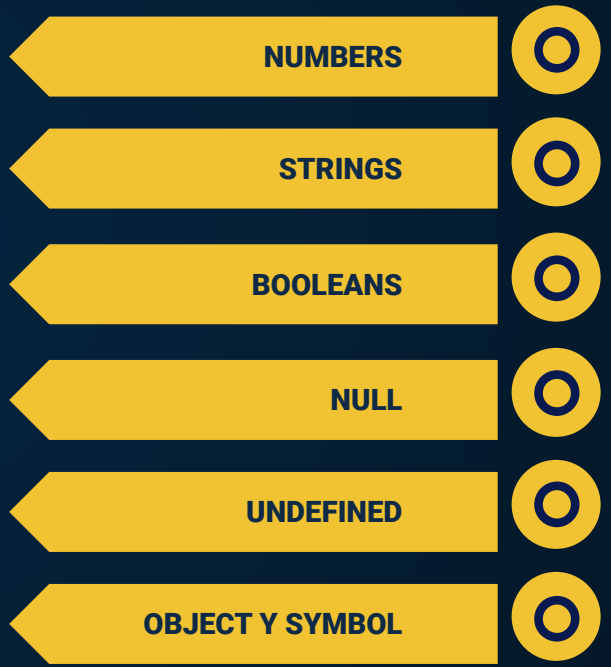
Código 4. Hola Mundo en JS con un Script Externo

02. TIPOS DE DATOS

TIPOS DE DATOS

Como en todo lenguaje de programación existen diferentes tipos de datos.

Vamos a conocer algunos de los tipos de datos más utilizados en JavaScript. Cubriremos en términos generales cada uno de ellos.



Victor Saldivia Vera | victor.saldivia@ulagos.cl

NUMBERS

```
1  let edad = 29;
2  let sueldo = 1200000;
3  const PI = 3.14;
```

El tipo number representa tanto números enteros como de punto flotante. En JavaScript, **no existe un tipo de dato específico para representar números enteros o números con decimales,** a diferencia de Python.

Al igual que en otros lenguajes se pueden realizar diferentes operaciones aritméticas como: suma, resta, multiplicación y división.

Además de los números comunes, existen los llamados **“valores numéricos especiales”** que también pertenecen a este tipo de datos: **Infinity**, **-Infinity** y **NaN**.

Victor Saldivia Vera | victor.saldivia@ulagos.cl

NUMBERS

(Notación Científica y BigInt)

Los números en notación científica se representan utilizando la notación exponencial. Se utiliza el formato **xEn**, donde **x es el coeficiente** y **n es el exponente**. El x puede ser un número decimal o entero, y el n puede ser positivo o negativo.

Por ejemplo, el número 1 millón en notación científica sería representado como **1e6** (como se muestra en la imagen de la izquierda).

En cuanto a los **BigInt**, se utilizan para representar enteros de tamaño muy grande, superando el límite de Number. Los BigInt se crean utilizando el **sufijo n** al final de un número entero.

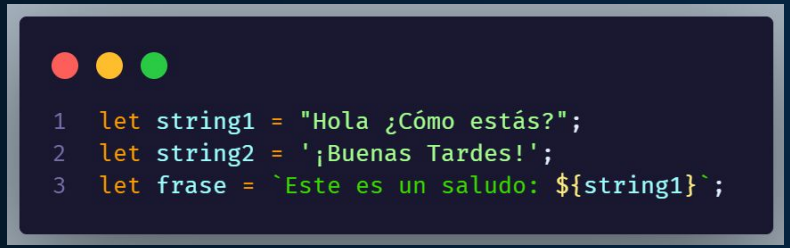
```
1  const n_grande = 1e6; // 1 millón
2  const n_pequeno = 1e-6; // 0.000001
```

```
1  const bigInt = 1234567890123456789012345678901234567890n;
```


Victor Saldivia Vera | victor.saldivia@ulagos.cl

STRINGS

Las cadenas de texto (strings) son un tipo de dato que representa una secuencia de caracteres. En JavaScript, podemos crear strings utilizando comillas simples o comillas dobles. También se puede ocupar backticks (comillas invertidas).



```
1  let string1 = "Hola ¿Cómo estás?";
2  let string2 = '¡Buenas Tardes!';
3  let frase = `Este es un saludo: ${string1}`;
```

Tanto el uso de comillas dobles o simples no tiene ninguna diferencia. Los backticks son comillas de "funcionalidad extendida". Nos permiten incrustar variables y expresiones en una cadena de caracteres encerrándolas en ${...}, como se muestra en el ejemplo de arriba.

ULAGOS

PRÓXIMA CLASE

TIPOS DE DATOS EN JS