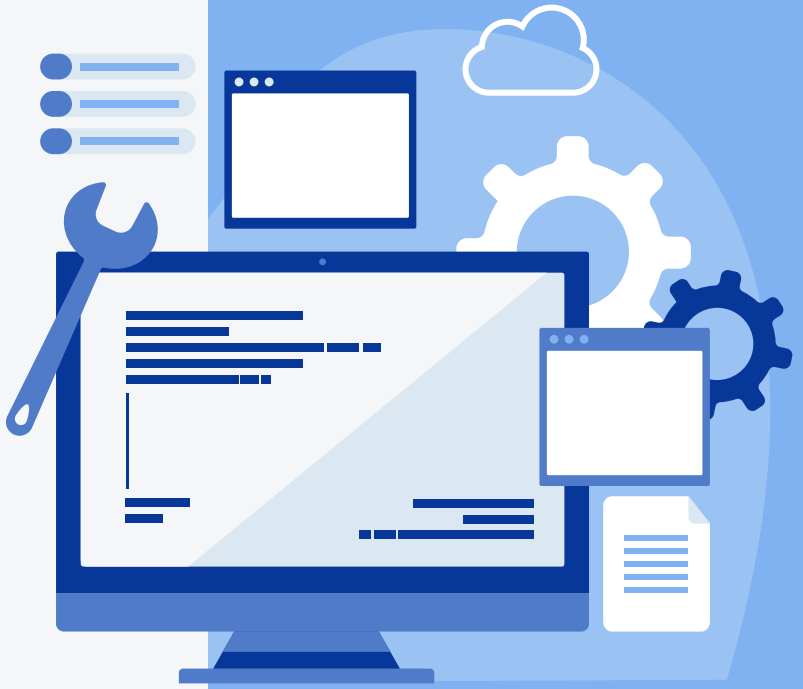
Introducción a **JavaScript**



Cordero Hernández, Marco R.

En sesiones pasadas...

- Flexbox
- Media Queries
- Bootstrap / CDN
- Componentes de Bootstrap
- Grid System

CONTENIDOS

01 – INTRODUCCIÓN

Introducción

JavaScript es un lenguaje de programación ligero, *interpretado*, y *orientado a objetos*. Es mejor conocido por ser el lenguaje de script(ing) para páginas web, sin embargo, hoy en día se ha extendido a aplicaciones más allá de los navegadores.

Es un lenguaje *multiplataforma* y *dinámico*.

Introducción

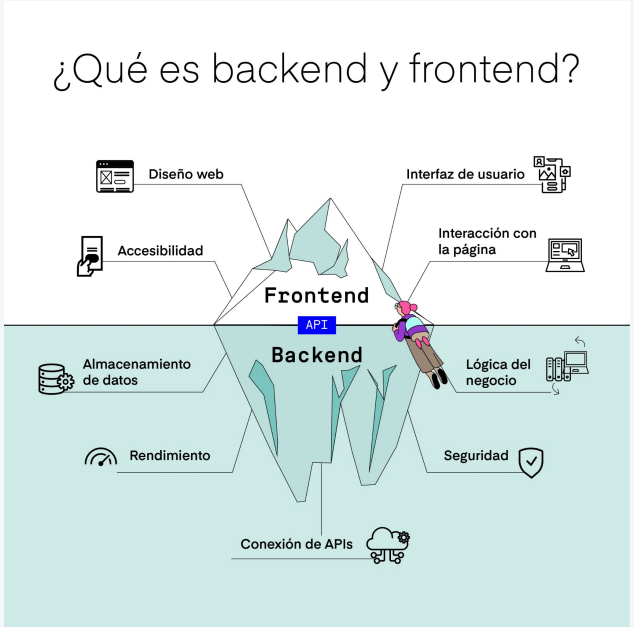
JavaScript puede ejecutarse donde sea que exista un *motor* que logre interpretarlo. Los navegadores principales implementan:

- Chrome - V8
- Firefox - SpiderMonkey
- Safari - JavaScriptCore
- Explorer - Chakra

Alcance de JavaScript

Las capacidades del lenguaje dependen del ambiente donde se ejecuten: *front-end* o *back-end*.

En un navegador, **JS** se encarga de manipular la página web, la interacción con el usuario *y la comunicación con el servidor*.



Alcance de JavaScript

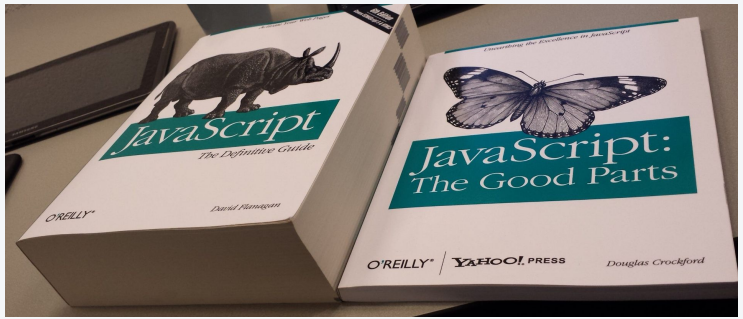
Adicional a lo anterior, **JS** puede realizar lo siguiente en un navegador:

- Añadir nuevo contenido, modificar contenido existente, y modificar estilos de un documento HTML
- Reaccionar a acciones de los usuarios, ejecutarse con acciones del mouse, teclas, etc.
- Enviar solicitudes sobre la red a servidores remotos, descargar y actualizar archivos
- Obtener y guardar *cookies**, hacer preguntas al usuario, mostrar mensajes
- Almacenar datos *del lado del cliente**

Alcance de JavaScript

Lo que **JS** no puede hacer en un navegador es:

- Lectura y escritura del disco duro, ejecución de programas; Ninguna acción que interactúe con el sistema operativo
- Manipular cámara y micrófono *sin permiso del usuario*
- Manipular el contenido de otras ventanas (cada una ejecuta en su “propio ambiente”)
- Ejecutarse en múltiples hilos o procesadores



ECMAScript

Es una especificación creada para estandarizar **JS**.

- La especificación con mayor soporte hasta el momento se le conoce como **ECMAScript 2015** o **ES6**, sin embargo, nuevas versiones han salido desde entonces
- A partir de **ES6**, la numeración es anual, por ejemplo: **ES7 -> 2016, ES11 -> 2020, 2024 -> ?**
- Si se desea utilizar sintaxis superior a **ES6** (la principal de este curso), se puede usar la herramienta Babel para “transpilar” código más reciente
- Tabla de compatibilidad de **ES6**.

01.1 – FUNDAMENTOS – USO/IMPORTACIÓN

Importar JavaScript en HTML

Al igual que CSS, JS puede invocarse en la cabeza o el cuerpo del documento.

```
<p>JS se ejecutó antes que esta línea</p>
<script>
   alert('Hola mundo');
</script>
<p>Cargado</p>
```

Importar JavaScript en HTML

Nuevamente, al igual que CSS, se pueden utilizar archivos externos

```
<!-- Archivo HTML -->
<script src="importacion.js"></script>


/* Archivo JS */
alert('Hola mundo');
```

01.1 – FUNDAMENTOS – LENGUAJE

Aspectos básicos

El punto y coma (;) no es necesario en JS, sin embargo, se recomienda su uso (puede forzarse con *“use strict”* al inicio de un archivo o dentro de una función).

```
// Comentarios de una línea
/* Comentarios de
  varias líneas */

// Variables
var x;              // Definición de variable
x = 38.5;           // Asignación de variable
var x = 33.333;     // Correcto

let y = 402;        // Variable de bloque
y = 1024;           // Correcto
let y = 2048;       // Incorrecto

const z = 'Texto'; // Declaración de constante
const w;            // Incorrecto
z = 'Texto 2';      // Incorrecto
```

Variables

Las variables deben contener solo letras, dígitos, o los símbolos $ y _

No deben iniciar con números

```
/* Variables */
// "use strict"

// Declaración inválida de
variables
let 1 asdf;
let vr - 1;
let
let = 1; // Tiempo de ejecución
let
return = 1;
otroVar = 1; // Error con "use
strict"
```

```
// Declaración válida de variables
let var1 = 'Texto';
let var2 = 10;
let var3 = "hola";
// Equivalente a let var1 =
'Texto', var2 = 10, var3 = "hola";
let var4 = var5 = var6 =
'múltiple';
let $var1;
let  var2;
let _;
let $;
let párrafo;
```

Tipos de datos

JS maneja los siguientes tipos de datos:

- Números (number)
- Cadenas de texto (string): entre comillas “” o ‘’
- Booleanos (boolean): true || false
- Array (array): Estructura que permite almacenar varios valores de cualquier tipo en una misma referencia [ ]
- Objeto (object): Cualquier cosa en JS es un objeto { }

Ejercicio rápido: Ejecuta 0.1 + 0.7 en tu navegador

Tipos de datos

```
// Declaración
let number1 = 3;
let number2 = 1.4556;
let str1 = "cadena1";
let str2 = 'cadena2';
let str3 = `cadena${number1}`;
let boolean1 = true;
let boolean2 = false; // !boolean1
let array1 = [1, 'texto', "otro",
1.34, true, ['celda0', 'celda1']];
let valArray = array1[0];
```

```
// Tipos de las variables en tiempo de
ejecución
console.log(typeof number1); // number
console.log(typeof number2); // number
console.log(typeof str1); // string
console.log(typeof str2); // string
console.log(typeof str3); // string
console.log(typeof boolean1); // boolean
console.log(typeof boolean2); // boolean
console.log(typeof array1); // object
console.log(typeof valArray); // number
```

Tipos de datos

```
// Reasignacion por tipado dinámico
let dato = "texto";
dato = 234; // Correcto
dato = true; // Correcto

// Números especiales
console.log(1 / 0); // Infinity
console.log(-1 / 0); // -Infinity
console.log('hola' / 2); // NaN
console.log(typeof Infinity); //
number
console.log(typeof - Infinity); //
number
console.log(typeof NaN); // number ¿?
```

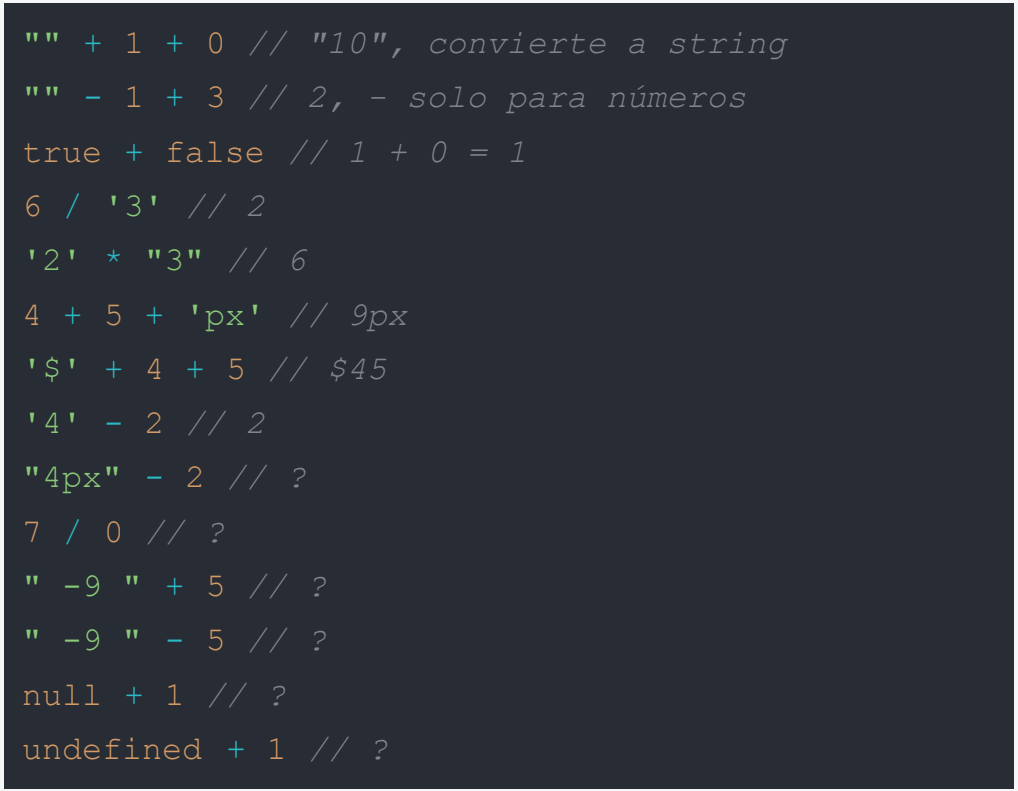
```
// Casos especiales de typeof
console.log(typeof undefined);
// undefined
console.log(typeof alert);
// function
console.log(typeof noExiste);
// undefined
console.log(typeof typeof
(noExiste)); // ?
console.log(typeof null); // ?
```

Conversiones de tipos

JS puede (intentar) convertir a sus datos primitivos:

- A cadena: String(valor)
- A número: Number(valor) o con operaciones para números (- /)
  - undefined -> NaN
  - null -> 0
  - true, false -> 0, 1
  - Cadena -> 0 si es vacía, número si solo hay valores numéricos (pueden haber espacios al inicio), o NaN
- A booleano: Boolean(valor)
  - Cadena vacía, 0, null, undefined, NaN -> false
  - Otros -> true

Conversiones de tipos



```
"" + 1 + 0 // "10", convierte a string
"" - 1 + 3 // 2, - solo para números
true + false // 1 + 0 = 1
6 / '3' // 2
'2' * "3" // 6
4 + 5 + 'px' // 9px
'$' + 4 + 5 // $45
'4' - 2 // 2
"4px" - 2 // ?
7 / 0 // ?
" -9 " + 5 // ?
" -9 " - 5 // ?
null + 1 // ?
undefined + 1 // ?
```

Operadores básicos

Similares a lenguajes comunes como C o Java:

- Suma, concatenación: +
- Resta, multiplicación y división: -, *, /
- Operador de asignación: =
- Módulo o residuo: %
- Exponenciación: **
- Incremento en 1: ++ (antes o después)
- Decremento en 1: – (antes o después)
- Operadores bitwise
  - AND &, OR |
  - XOR ^, NOT ~
  - LEFT SHIFT <<, RIGHT SHIFT >>
  - ZERO-FILL RIGHT SHIFT >>>

```
console.log(number1+number2); // Suma
console.log(str1+str2); // Concatena
console.log(number1-number2); // Resta
console.log(number1*2.5); // Multiplica
console.log(number1/number2); // Divide
console.log(number1=number2); // Asigna a
number1
console.log(2**3) // 2*2*2 = 8
console.log(number1++); // ?
console.log(++number1); // ?
console.log(number1--); // ?

console.log(4 & 2); // 0
console.log(4 | 2); // 6
console.log(6 ^ 2); // 4
console.log(~1); // -2
```

Operadores comparativos y relacionales

- Igualdad ==
- Igualdad *estricta* ===
- Negación !
- Distinto !=
- Distinto *estricto* !==
- Relacionales >, <, >=, <=

- Para cadenas, compara letra por letra
- Con relacionales, convierte a valores numéricos

```
console.log(1 == 1);                    // true
console.log("1" == 1);                  // valor
:= true
console.log("1" === 1);                 // ?
console.log("hola" == "hola");          // true
console.log(1 == true);                 // ?
console.log(0 == false);                // ?
console.log(1 === true);                // ?
console.log(0 === false);               // ?
console.log(!true);                     // ?
console.log(1 != "1");                  // false
console.log(1 !== "1");                 // ?
console.log("hola" !== "hola");         // false
console.log(false == '');               // true
console.log(null == undefined);         // true
console.log(null === undefined);        // ?
console.log(null > 0);                  // ?
console.log(null == 0);                 // ?
console.log(null >= 0);                 // ?
console.log(undefined > 0);             // ?
console.log(undefined == 0);            // ?
console.log(undefined >= 0);            // ?
```

Consideraciones sobre NaN

NaN no es igual a nada, por ende, tampoco es igual a **NaN** (es incluso menos específico que *null*).

Para verificar si un valor evalúa a **NaN**, existe la función *isNaN()*.

```
NaN == NaN // false
NaN === NaN // false
Number('hola') == NaN // false
isNaN('hola') // true
```

Otros operadores

Los siguientes operadores pueden usarse en la asignación:

- +=, *=, /=, -=, &&=, ||=

Pueden haber múltiples asignaciones por comas, pero solo la última evaluación será considerada

```
// Asignación
var x = 27;
x += (8 + 5);
x *= 4;
x /= 2;
x -= 11;
console.log(x); // ?

// Lógicos
var y = false;
x ||= y; // valor de x
x &&= y; // ?
x != x; // ?

// Evaluaciones separados por comas
let a = (1 + 2, 3 + 4); // 7
let b, c = (1 + 5, 2 + 9); // ?

for (a = 1, b = 3, c = a * b; a < 10; a++) {}
```

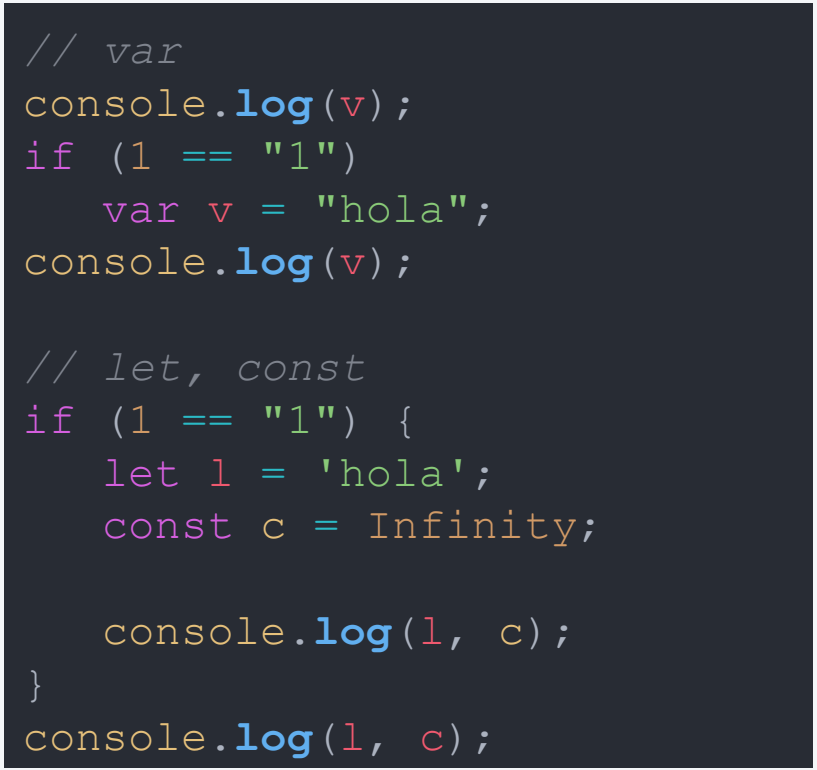
var* vs *let* y *const

Previamente, la palabra reservada ***var*** era la única manera de declarar variables, ahora, con la introducción de **ES6**, se puede hacer uso de ***let*** *y se recomienda su uso cuando sea posible*.

Las variables declaradas con ***var*** no tienen límite de scope, las cuales pueden verse como globales o procesadas al inicio de las funciones. Son visibles entre bloques de código.

Las variables declaradas con ***let*** y ***const*** son válidas únicamente dentro del bloque donde se encuentran y después de que se declaran.

var vs *let* y *const*



```
// var
console.log(v);
if (1 == "1")
   var v = "hola";
console.log(v);

// let, const
if (1 == "1") {
   let l = 'hola';
   const c = Infinity;

   console.log(l, c);
}
console.log(l, c);
```

Condicionales

```
// if
if (condicion) {
    console.log(true);
}

if (condicion)
    console.log(true);

if (condicion) console.log(true);

// if-else
if (condicion) {
    console.log(true);
} else {
    console.log(false);
}

if (condicion) console.log(true);
else console.log(true);
```

```
// if-else if-else
if (condicion) console.log('Inicial');
else if (condicion2) console.log('Segundo');
else console.log('Alternativa');

// Ternario
console.log(condicion ? 'Válido' : 'Inválido');
let x = (cond1 & cond2) ? true :
    (cond1 & cond3) ? 'segundo' : false;


// Switch
switch (variable) {
    case 1:
        ...
        break;
    case 2:
        ...;
        break;
    case 'algo_más':
        ...x;
        break;
    default:
        ...
}
```

Ciclos while/for

```
// while
while (cond1 || cond2) {
    ...

    if (cond1 == 20 && cond2 == -1) continue;
    if (cond1 == 20) break;
}

// do-while
let i = 0;
do {
    console.log(i);
} while (++i < 10);
```

```
// for -> inicio; condiciones; paso
for (let i = 20; i != 0; i--) {
    ...
}

// Romper un ciclo específico
loop1: while (true) {
    loop2: while (--i) {
        if (i == 2) break loop1;
    }
}
```

Números

```
let num1 = 1e4;                     // 10 000
let num2 = 1e-3                     // 0.001
let hex1 = 0xFF;                    // 255
hex1 = 0xff;                        // 255
let bin = 0b1010101;                // 85
let octal = 0o11;                   // 9
console.log(hex1.toString(16));     // ff
console.log(hex1.toString(2));      // 11111111

Math.random();                      // Aleatorio
Math.max(-10, 20, 5, 1, 255);       // 255
Math.min(-10, 20, 5, 1, 255);       // -10
Math.floor(3.1);                    // 3
Math.floor(-2.1);                   // -3
Math.ceil(3.1);                     // 4
Math.ceil(-2.1);                    // -2
Math.round(3.5);                    // 4
Math.trunc(3.8);                    // 3

let num3 = 1.3819;
console.log(num3.toFixed(2));       // 1.38
console.log(parseInt('100px'));     // 100
console.log(parseFloat('2.3em'));   // 2.3
console.log(parseInt('2.3'));       // 2
console.log(parseFloat('2.3.4'));   // 2.3
```

Cadenas (Strings)

```
let str = 'Hola mundo ';
console.log(str.length); // Longitud = 11
console.log(str[0]); // H
console.log(str.charAt(4)); // ' '
console.log(str[12]); // undefined
console.log(str.charAt(12)); // ''
console.log(str[-1]); // ?

for (let c of str)
   console.log(c);

str[0] = 'C'; // ?
console.log(str.toUpperCase()); // HOLA MUNDO
console.log(str.toLowerCase()); // hola mundo
console.log(str.trim()); // Hola mundo\0
```

02 – FUNCIONES

Concepto y declaración

Como en la mayoría de lenguajes de programación, **JS** soporta la declaración de bloques de código reusables e invocables definidos por el usuario, los cuales aceptan argumentos de longitud variable... es decir, *funciones*.

```
// Declaración de una función
function saludarA(nombre) {
   console.log(`Hola ${nombre}`);
}

// Invocación de una función
saludarA('Chuyita');
```

Consideraciones

Una función que no devuelve nada (sin *return* o *return* sin valor de retorno) regresa *undefined* (lo cual no es un comportamiento erróneo).

Algo adicional para considerar es que el valor de retorno de una función (en caso de haberlo) *debe estar en la misma línea que el mismo* ***return***, ya que un salto de línea significa el fin de la sentencia.

También, es posible definir parámetros con valores por default (param1 = false).

Consideraciones

```
// Función con parámetros predefinidos
function pre(param1 = true, param2 = false) {
    if (param1)
        console.log('Parámetro 1 definido');

    if (param2)
        console.log('Párametro 2 definido');
}
pre(false, true);

// ¿Qué regresaría?
let v3 = 3;

function scopout(v1, v2) {
    return v1 + v2 + v3;
}
```

02.1 - EXPRESIONES

Expresiones de funciones

Las funciones pueden verse como un valor que representa una acción, por ende, pueden ser almacenadas en variables y ejecutadas desde las mismas.

```
let mensaje = function mostrarMensaje(texto) {
    console.log(texto);
};
mensaje('Hola');
console.log(mensaje); // Muestra el código ¿?
console.log(typeof mensaje); // function
```

02.2 – CALLBACKS

Callbacks

Las funciones pueden ser pasadas como argumentos de otras funciones, lo cual permite simular un poco del comportamiento de programación funcional.

```
function crearNuevoUsuario(id, usuario, guardarDatos) {
   if (id > 0 && id < 10_000) {
       return guardarDatos(id, usuario);
   }

   console.log(`ID incorrecto (${id})`);
}
let base1 = function (id, usuario) { // Función anónima
   console.log(`Usuario ${usuario} guardado en base normal`);
};
let base2 = function(ID, username) { // Función anónima
   console.log(`User with ID ${ID} saved as ${username}`);
};

crearNuevoUsuario(1, 'master chief', base1);
crearNuevoUsuario(100, 'mf_d00m', base2);
```

Ejercicio

Crear una función llamada generarReporte, que reciba nombre y la calificación, y dos funciones: fnAprobado y fnReprobado, las cuales se ejecutarán en caso de que la calificación sea aprobatoria o reprobatoria respectivamente, estas últimas 2 funciones reciben como argumento el nombre (solo muestran algún mensaje con el nombre del alumno y la acción a aplicar).

Mandar a llamar esta función de las siguientes formas:

- Creando previamente las funciones
- Guardando las funciones en variables
- Directamente en la llamada (funciones anónimas)

02.3 - ARROW FUNCTIONS

Concepto

Otra sintaxis para declarar funciones privadas dentro de variables es usando expresiones de flechas (arrow functions).

```
/*
   let func = (arg1, arg2, ..., argN) => expresion;
   --- EQUIVALE A ---
   let func = function(arg1, arg2, ..., argN) {
       return expresion;
   }
*/
let suma = (a, b) => a + b;
console.log(suma(20, 30)); // ?

// Si la función recibe un solo argumento, pueden omitirse los paréntesis
let doble = n => n * 2;
console.log(doble(210)); // ?

let saludar = () => console.log('Hola mundo');
saludar();

let concat = (a, b) => {
   let resultado = '';
   resultado += a + b;
   return resultado;
}
console.log(concat('Adiós', 'mundo')); // ?
```