¿Como surgió JavaScript?

JavaScript nace con la necesidad de generar dinamismo en las páginas web y que a su vez los usuarios y las empresas pudieran interactuar unos con otros.

¿Por qué aprender JavaScript?

1- JavaScript es un lenguaje muy versátil, además tiene una comunidad enorme de desarrolladores.

2- Está presente en todos lados:

1. Entorno web (Frontend)
2. Aplicaciones Android y IOS
3. Aplicaciones de escritorio
4. Backend

3- Se trabaja con gran variedad de herramientas

1. Framework - frontend: React, Vue, Angular
2. Framework - Mobile: React native (Android, IOS)
3. Framework - desktop: Electron (Windows, MacOS)
4. Entorno de ejecución / backend y IOT (Internet of things): Node.js

Sitios web basados en:

1. Angular: Forbes
2. React: Airbnb
3. Vue: GitLab

Aplicaciones basadas en React Native:

1. Uber Eats
2. Discord
3. Instagram

Aplicaciones para Escritorio basados en Electron:

1. Visual Studio Code
2. WhatsApp
3. Twitch

Compañías que usan Node.JS para parte de su backend:

1. Netflix
2. LinkedIn
3. PayPal

¿Qué elementos trabaja JavaScript?

- Datos (que se guardan en memoria), y se tipean.
- Tareas(funciones), acciones y procesos que realizamos usando los datos guardados previamente.

¿Cuáles son los tipos de datos?

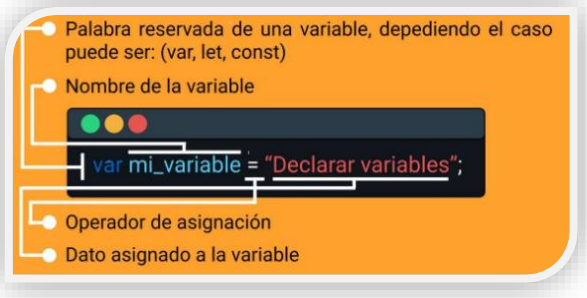
Tipos de datos: string, boolean, number (decimal, entero), object, undefined.

Para saber qué tipo de dato es un valor, podemos usar la función typeof seguido de un valor o variable.

```
> typeof 10
< 'number'
> typeof "diego"
< 'string'
> typeof {"diego":10}
< 'object'
> typeof [1,2,3,4,5]
< 'object'
> typeof diego
< 'undefined'
> typeof true
< 'boolean'
```

¿Cómo declarar variables?

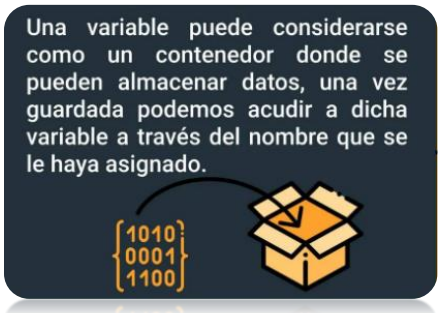
Dentro de JavaScript tenemos tres formas de declarar una variable las cuales son: **var**, **const** y **let.**

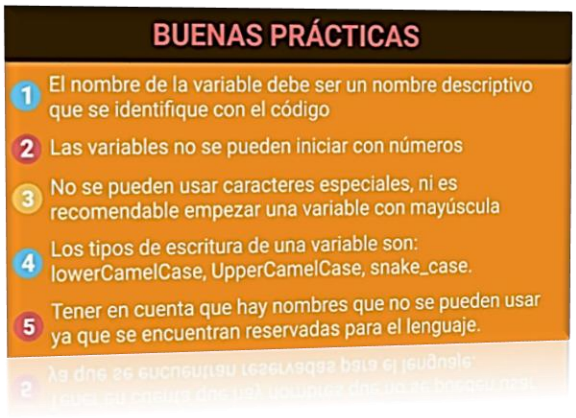


- **Var:** Era la forma en que se declaraban las variables hasta ECMAScript 5. Casi ya no se usa.

A partir de ECMAScript 6

- **Let:** variables que pueden ser modificadas o reasignadas.
  Ej: let nombre = "Diego"





- **const:** variables que nunca van a ser modificadas.
No se pueden reinicializar, es una constante única, por ende, no puede haber otra inicializada con el mismo nombre.
No se pude reasignar, una vez inicializada es inmutable, es decir no puede cambiar.

Ej: const activo = true

¿Qué son Operadores?

En JavaScript existen símbolos conocidos como operadores, los cuales son necesarios para realizar diferentes tipos de operaciones; por ejemplo, el signo = se utiliza para asignar un valor.
Los principales tipos de operadores en JavaScript son:

- Operadores Aritméticos: se utilizan para efectuar operaciones matemáticas básicas.
    - + (Suma)
    - – (Resta)
    - * (Multiplicación)
    - / (División)

- Operadores de Comparación: Se usan para comparar dos valores y determinar si son iguales o no.
    - == (igual) - devuelve verdadero si los valores son iguales
    - === (estrictamente igual) - devuelve verdadero si los valores son iguales y del mismo tipo
    - != (diferente) - devuelve verdadero si los valores son diferentes
    - !== (estrictamente diferente) - devuelve verdadero si los valores son diferentes o de diferente tipo
    - > (mayor que) - devuelve verdadero si el valor de la izquierda es mayor que el valor de la derecha
    - < (menor que) - devuelve verdadero si el valor de la izquierda es menor que el valor de la derecha
    - >= (mayor o igual que) - devuelve verdadero si el valor de la izquierda es mayor o igual al valor de la derecha
    - <= (menor o igual que) - devuelve verdadero si el valor de la izquierda es menor o igual al valor de la derecha.

- Operadores Lógicos: Combinan expresiones lógicas y determinar su verdad o falsedad.
    - && (AND / Y)- devuelve verdadero sólo si ambas expresiones son verdaderas.
    - || (OR / O)- devuelve verdadero si al menos una de las expresiones es verdadera.
    - ! (NOT / Negado)- invierte el valor lógico de una expresión, convirtiendo verdadero en falso y viceversa.

- Operadores de asignación:
    - = (Asignar): var numero = 5
    - += (sumar y asignar): numero += 1; (Equivale a decir numero = numero + 5)
    - -= (restar y asignar): numero -= 1; (Equivale a decir numero = numero - 5)

- *= (multiplicar y asignar): numero *= 1; (Equivale a decir numero = numero * 5)
- /= (dividir y asignar): numero /= 1; (Equivale a decir numero = numero / 5)
- %= (módulo y asignar): numero %= 1; (Equivale a decir numero = numero % 5)
- **= (potencia y asignar): numero += 1; (Equivale a decir numero = numero ** 5)

¿Qué son Condicionales?

Una expresión condicional es un conjunto de instrucciones que evalúan una expresión

y solo se ejecutarán si dicha expresión es verdadera.

JavaScript admite dos expresiones condicionales: (if...else) y (switch).

- **If-else:** La expresión if ejecuta una instrucción si una condición lógica es true.
  de lo contrario else para ejecuta una instrucción si la condición es false.

```
if (numero1 > Numero2){
    alert (`El numero ${numero1} es mayor que ${Numero2}`);
} else {
    alert (`El numero ${numero1} es menor que ${Numero2}`);
}
```

  - **else if:** realiza una validación adicional en caso de que la validación previa resultara en false.

```
if (numero1 > Numero2){
    alert(`${numero1} es mayor que ${Numero2}`);
} else if (numero1 == Numero2){
    alert(`${numero1} es igual que ${Numero2}`);
} else {
    alert(`${numero1} es menor que ${Numero2}`);
}
```

- **Switch:** Permite ejecutas un bloque de código basándose en casos.
  La sintaxis de switch consiste en una expresión a evaluar y en base a los resultados esperados, se desarrollan los casos a ejecutar, adicional se coloca break para evitar que se ejecuten los demás casos y default para una respuesta por defecto en caso de que ningún caso se cumpla.

Ejemplo:

```
let producto = "carne";
switch(producto){
    case "carne":
        alert("¡Compraste Carne!");
        break;
    case "verduras":
        alert("¡Compraste Verduras!");
        break;
    default:
        alert("Opción no valida.");
}
// La salida será: ¡Compraste Carne!
// ya que se cumple el primer caso.
```

¿Qué son Arrays?

Un array es una lista que contiene una colección ordenada de valores.

- Cada valor se almacena en una posición específica dentro del array, y se puede acceder a estos mediante un índice numérico que determina su posición.
- Los arrays son muy versátiles y se utilizan para almacenar y manipular datos de varios tipos, como números, cadenas, objetos, etc.

```
let miArray = [1,2,3,"uno","dos","tres",[1,2,"uno","dos"]];
```

- Se pueden crear arrays vacíos o con elementos iniciales, y se pueden modificar y acceder a los elementos de un array mediante métodos y propiedades específicas.

```
let colores = ["amarillo", "azul", "rojo"];
```

- length: Retorna la longitud de un array.
- push: Agrega elementos al final de un array, recibe como parámetros los elementos a agregar. Ejemplo: colores.push("Blanco", "Negro");
- unshift: Agrega elementos al inicio de un array, recibe como parámetros los elementos a agregar. Ejemplo: colores.unshift("Gris", "Plateado");
- pop: Elimina el elemento al final de un array, No recibe parámetros.
  Ejemplo: colores.pop();
- shift: Elimina el elemento al inicio de un array, No recibe parámetros.
  Ejemplo: colores.shift()

¿Qué son las Funciones?

Las funciones son las tareas que va a llevar a cabo el navegador.

Existen 2 tipos de funciones:

- **Funciones Declarativas**:
  En las funciones declarativas, utilizamos la palabra reservada function al inicio para poder declarar la función:

```
function saludar(nombre) {
  console.log(`Hola ${nombre}`);
}

saludar('Diego');
```

- **Funciones de expresión**:
  En la función de expresión, la declaración se inicia con la palabra reservada var, donde se generará una variable que guardará una función anónima:

```
var nombre = function(nombre){
    console.log(`Hola ${nombre}`)
}

nombre('Diego');
```

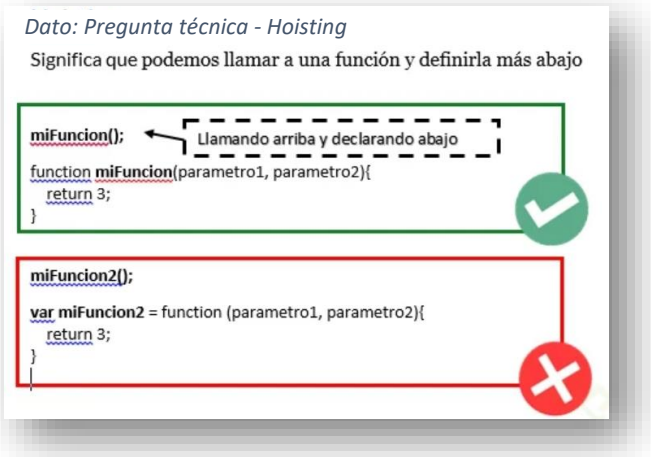
  En la función de expresión, la función podría o no llevar nombre, aunque es más común que se hagan anónimas.

Ambas pueden llevar parámetros, estos son datos que se procesaran, cada parámetro va separado por una coma y cada instrucción que tenga la función debe terminar con ; .

- **Diferencias: A** las funciones declarativas se les aplica Hoisting, y a la función de expresión, no.

  Ya que el Hoisting solo se aplica en las palabras reservadas var y function.

  Lo que quiere decir que, con las funciones declarativas, podemos mandar a llamar la función antes de que ésta sea declarada, y con la expresión de función, no, tendríamos que declararla primero, y después mandarla llamar.



¿Qué son Objetos?

Un objeto es una estructura de datos que contiene una colección de propiedades y métodos. Cada propiedad es un par clave-valor que representa una característica o atributo del objeto, y cada método es una función que representa una acción o comportamiento del objeto.

Un ejemplo simple de un objeto es un objeto persona:

```
let persona = {
    nombre: "Daniel",
    edad: 30,
    saludar: function() {
        console.log(`Hola, mi nombre es ${this.nombre}`);

    }
};

  console.log(persona.nombre);  // imprime "Daniel"
  console.log(persona.edad);     // imprime 30
  persona.saludar();             // imprime "Hola, mi nombre es Daniel"
```

¿Qué son Loops?

Los loops o bucles son estructuras de control de flujo en programación, que permiten repetir un bloque de código un número determinado de veces o mientras se cumpla una condición.
Esto es útil cuando se necesita ejecutar una tarea repetitiva varias veces, por ejemplo, para recorrer una lista de elementos y realizar una acción en cada uno de ellos.

Existen dos tipos de loops principales: **for** y **while.**

- **for:** Es un loop estructurado que se utiliza para repetir un bloque de código un número fijo de veces.

  Tiene tres partes: la inicialización, la condición y el incremento.

  ```
  for (i = 1; i <= 5; i++) {
      console.log(i);
  }
  ```

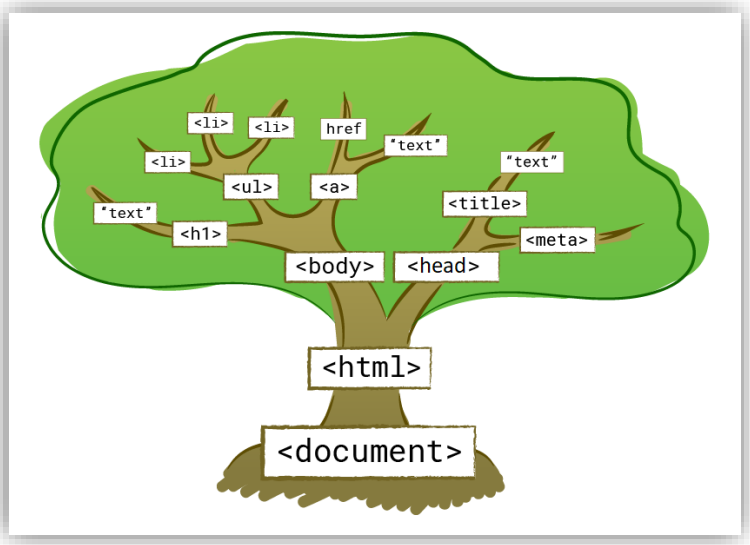
- **while:** es un loop condicional que se utiliza para repetir un bloque de código mientras se cumpla una condición. La condición se evalúa al principio de cada iteración y, si es verdadera, se ejecuta el bloque de código, luego la condición se vuelve a evaluar, y el loop continúa repitiéndose hasta que la condición sea falsa.

```
while (i <= 5) {
    console.log(i);
    i++;
}
```

¿Qué es el DOM?

El "DOM" (Modelo de Objeto del Documento), también conocido como árbol de elementos, es una representación en forma de árbol de la estructura de un documento HTML.
Cada nodo en el árbol de elementos representa un elemento HTML en la página web.

El árbol de elementos comienza con el elemento raíz, que es "document", y luego se descompone en elementos hijo, nietos, bisnietos, etc. Cada elemento tiene propiedades, como id, class, name, value, etc... que pueden ser accedidas mediante selectores y manipuladas con JavaScript.



¿Qué son Selectores?

Son herramientas que permiten seleccionar elementos del DOM (elementos HTML) para manipularlos en una página web.

Los más comunes son:

- .getElementById(): selecciona un elemento con un id específico.
- .getElementsByNames(): selecciona elementos con un name especifico.
- .getElementsByClassName(): selecciona elementos con una clase específica.
- .getElementsByTagName(): selecciona elementos con un nombre de etiqueta específico.

¿Qué son Eventos?

Los eventos son la manera en la cual JavaScript controla las acciones de los visitantes y define el comportamiento de la página cuando estos se produzcan.

Para entender los eventos es necesario conocer algunos conceptos básicos:

- **Evento:** Es algo que ocurre. Generalmente los eventos ocurren cuando el usuario interactúa con el documento, pero podrían producirse por situaciones ajenas al usuario, como el caso en el que una imagen no este disponible.

- **Tipo de evento**: Es el tipo del suceso que ha ocurrido, por ejemplo, un clic sobre un botón o el envío de un formulario. Cada tipo de elemento de la página ofrece diversos tipos de eventos de JavaScript.

- **Manejador de evento**: Es el comportamiento que nosotros podemos asignar como respuesta a un evento. Se especifica mediante una función en JavaScript, que se asocia a un tipo de evento en concreto. Una vez asociado el manejador a un tipo de evento sobre un elemento de la página, y cada vez que ocurre ese tipo de evento sobre ese elemento en concreto, se ejecutará el manejador de evento asociado.

Formas de manejar eventos:

- **Mediante atributos HTML:** <button onClick="miFuncion ()">Presiona Aquí! </button>

- **Mediante propiedades JavaScript:** elemento.onClick = miFuncion (){ …código… }

- **Mediante addEventListener():** la forma más recomendable es hacer uso del método .addEventListener(), el cuál es mucho más potente y versatil para la mayoría de los casos.

  Ejemplo:

```
button.addEventListener("click", function() {
    alert("Hello!");
});
```

¿Qué Son Expresiones Regulares?

Las expresiones regulares(también conocida como regexp o regex) son una secuencia de caracteres que forman un patrón de búsqueda, y se utilizan para hacer comparaciones con texto. Son una herramienta poderosa para manipular y validar texto, y se pueden utilizar en conjunto con métodos de JavaScript como .match(), .replace() etc..

Se utilizan en muchos lenguajes de programación, incluyendo JavaScript, para verificar si una cadena de texto cumple con un determinado patrón o para reemplazar texto por un patrón determinado.
Las expresiones regulares están escritas entre barras / y pueden incluir una combinación de caracteres literales y caracteres especiales que representan un conjunto de caracteres posibles.

> Por ejemplo, la expresión regular /\d{3}-\d{3}-\d{2}-\d{2}/ busca un número de celular en formato 111-111-11-11.

Se pueden usar en una gran variedad de tareas, como validación de formularios, extracción de información de un párrafo etc...

Sintaxis:

```
const regexp = /palabra/i
const regexp2 = /p...bra/gi
const regexp3 = /.al..ra/gim
const regexp4 = /\D/gi
```

Coincidencias Básicas:

- . - Cualquier Caracter, excepto nueva línea
- \ - Indica que el siguiente caracter se debe tratar de manera especial o "escaparse".
- \d - Cualquier Dígitos (0-9)
- \D - No es un Digito (0-9)
- \w - Caracter de Palabra (a-z, A-Z, 0-9, _)
- \W - No es un Caracter de Palabra.
- \s - Espacios de cualquier tipo. (espacio, tab, nueva línea)
- \S - No es un Espacio, Tab o nueva línea.

Limites:

- \b - Limite de Palabra
- \B - No es un Límite de Palabra
- ^ - Inicio de una cadena de texto
- $ - Final de una cadena de texto

Cuantificadores:

- - Coincide con 0 o más ocurrencias del patrón anterior.
- + - Coincide con 1 o más ocurrencias del patrón anterior.
- ? - Coincide con 0 o 1 ocurrencia del patrón anterior.
- {3} - Número Exacto
- {3,4} - Rango de Números (Mínimo, Máximo)

Conjuntos de Caracteres:

- [] - Caracteres dentro de los brackets
- [^] - Caracteres que NO ESTAN dentro de los brackets

Grupos:

- ( ) - Grupo
- | - Uno u otro

probar en: http://regexr.com/77g8v

¿Qué Son Clases?

Una clase es una plantilla o molde para crear objetos(instancias), que tienen propiedades y métodos en común.

Se definen utilizando la palabra clave "class" seguida del nombre de la clase, y las llaves, dentro de las llaves se definen las propiedades y métodos de la misma.

Se componen internamente por:

- **Constructo**r: es un método especial de la clase que se ejecuta automáticamente cuando se crea una nueva instancia, su rol principal es inicializar los atributos de la clase con los valores pasados como argumentos al crear una nueva instancia, usando la palabra reservada new.

- **Método**: es una función (puede ser opcional) que retorna (con return) lógica. El método Saludo está fuera del scope del constructor, pero puede acceder a los atributos de la clase que se han declarado en el constructor a través de la palabra clave this. Esto es posible porque this se refiere a la instancia actual de la clase.

- **this**: hace referencia al elemento padre, es decir al objeto instanciado.

NOTA: *Es una convención común en JavaScript declarar las clases con la primera letra de cada palabra en mayúscula. Ejemplo "Persona" o "Usuario", sin embargo, es solo una buena práctica, no evita el buen funcionamiento de las clases.*

Esto se realiza con la finalidad de diferenciar de manera más fácil cuando se emplea una clase, una variable o una función.

Ejemplo:

```
class Persona {
    //Constructor: asigna valor a las propiedades.
    constructor(nombrePersona, edadPersona) {
        this.nombre = nombrePersona;
        this.edad = edadPersona;
    }
    //Método: retorna lógica.
    saludo() {
        return (`Hola, Me llamo ${this.nombre} y tengo ${this.edad} años.`);
    }
}
```

¿Qué es Try-Catch?

Es una estructura de control de excepciones que permite probar código para detectar errores y, en caso de que se produzca una excepción, ejecutar un bloque de código de manejo de excepciones para manejar el error.

El bloque "try" contiene el código que se está probando para errores, y el bloque "catch" contiene el código que se ejecutará en caso de que se produzca una excepción.

¿Qué es una Excepción?

Una excepción es un evento anormal que ocurre durante la ejecución de un programa que interrumpe el flujo normal de ejecución y puede ser tratado por el programador para prevenir errores graves.

Esto permite la continuidad ininterrumpida en la ejecución del código en caso de existir algún error en el proceso.

Finally: es una cláusula opcional, no es una propiedad ni un método como tal, sino una parte de la sintaxis de la estructura de control Try-Catch. Su propósito es permitir que los programadores ejecuten un bloque de código de limpieza o de seguimiento, registrar información después de que se haya ejecutado el bloque try y el bloque catch.

Excepciones comunes:

- ReferenceError: Se produce cuando se hace referencia a una variable no definida.
- SyntaxError: Se produce cuando hay un error de sintaxis en el código.
- TypeError: Se produce cuando un operador o función es invocado en un objeto que no es válido para ese tipo de operación o función.

Ejemplo:

```
try {
    console.log(5 / 0);
}catch (error) {
    console.log(`Error: No se puede dividir por cero. ${error}`);
}finally {
    console.log("Este código se ejecutará siempre al final");
}
```

¿Qué es el asincronismo?

El asincronismo es una técnica que permite a los programas ejecutar tareas de manera no bloqueante. Esto permite que el programa puede continuar con otras tareas mientras se completa una tarea asíncrona, en lugar de bloquearse y esperar a que se complete.

El asincronismo es fundamental para mejorar la experiencia del usuario y mejorar la eficiencia del programa, especialmente cuando se trata de tareas que pueden tardar un tiempo en completarse, como las solicitudes de red y consumo de API´s.

Hay varias formas de lograr el asincronismo, como promesas, async-await, setTimeout, setInterval y la utilización de eventos, cada uno de estos enfoques ofrece diferentes ventajas y desventajas y puede ser apropiado en función del uso que se le quiera dar.

En resumen, el asincronismo es una técnica esencial para mejorar la experiencia del usuario y la eficiencia del programa.

¿Qué son Callbacks?

Un callback es una función que se pasa a otra función como argumento y se invoca o inicia después de que algún evento o proceso ha terminado, de allí el nombre callback que traducido al español significa: llamar de vuelta.

En pocas palabras, la función receptora recibe mediante parámetros una función que viene definida y es iniciada/llamada dentro la misma función receptora.

Ejemplo:

```
function agregar(miArray, miCallback){
    miArray.push("Diego");
    miCallback();
}

let nombres = ["Jean","Carlos","Jose"];

agregar(nombres, function(){
        console.log(`Agregue un nombre al array`);
        console.log(nombres);
    }
);
```

¿Qué son Promesas?

Las promesas son una forma de manejar la asincronía.
Una promesa representa el resultado de una operación que aún no ha finalizado, pero que eventualmente producirá un resultado o un error.

Una promesa se crea usando la palabra clave new y la función constructora Promise.

La función constructora recibe como argumento una función que acepta dos callbacks, resolve y reject.
La función que se pasa a la promesa debe realizar la operación asíncrona y, una vez terminada, llamar a resolve si todo fue bien o a reject si ocurrió algún error.

```
const miPromesa =  new Promise( (resolve,reject)=> { ... });
```

Una vez creada la promesa, se pueden suscribir a ella para los métodos .then y .catch. para recibir su resultado o su error.

- .then : Es un método que recibe como argumento una función que será llamada cuando la promesa se resuelva (es decir, cuando se llame a resolve) y se le pasará como argumento el resultado de la operación asíncrona.

- .catch : Es un método que recibe como argumento una función que será llamada cuando la promesa sea rechazada (es decir, cuando se llame a reject) y se le pasará como argumento el error producido.

En resumen, las promesas permiten manejar de forma sencilla la asincronía, permitiendo encadenar operaciones y controlar los errores de manera eficiente.

Glosario:

- **Angular, Vue, React** son algunos de los frameworks que podemos utilizar para hacer productos web.

- **React Native**: framework para poder construir aplicaciones nativas para Android y IOS.

- **Electron:** framework que permite desarrollar aplicaciones para escritorio, tanto para MacOS como para Windows.

- **Frontend:** Parte visual, todo lo que se ve en nuestra web o App y con lo que podemos interactuar.

- **Backend:** Parte lógica, manejando las bases de datos, las interacciones y peticiones que el Frontend le va a pedir.

- **Node.JS**: Entorno de ejecución de JavaScript que corre en el Backend.
  También permite trabajar aplicaciones IOT (Internet of things/ Internet de las cosas), lo que hace inteligente ciertos dispositivos conectados a internet.

- **Coerción**: Consiste en cambiar de un tipo de dato a otro.
  - Coerción Implícita: cuando el lenguaje ayuda.
  - Coerción Explicita: cuando se fuerza la conversión.
    (Ej: convertir valor numérico (string) de un input a Number)