



Presentado Por:
Luis Somoza

FUNDAMENTOS DE: **JAVASCRIPT**



¿QUÉ ES
JAVASCRIPT?

A yellow square containing the letters 'JS' in a bold, black, sans-serif font, representing the JavaScript logo.

JS

¿QUÉ ES JAVASCRIPT?.js

En los años 90 todo era
estático (**HTML y CSS**),
haciendo todo aburrido y
tedioso.

Es un lenguaje de programación
orientado a la creación de
páginas y aplicaciones web
dinámicas.



¿POR QUÉ DEBERÍA DE
USAR **JAVASCRIPT**?



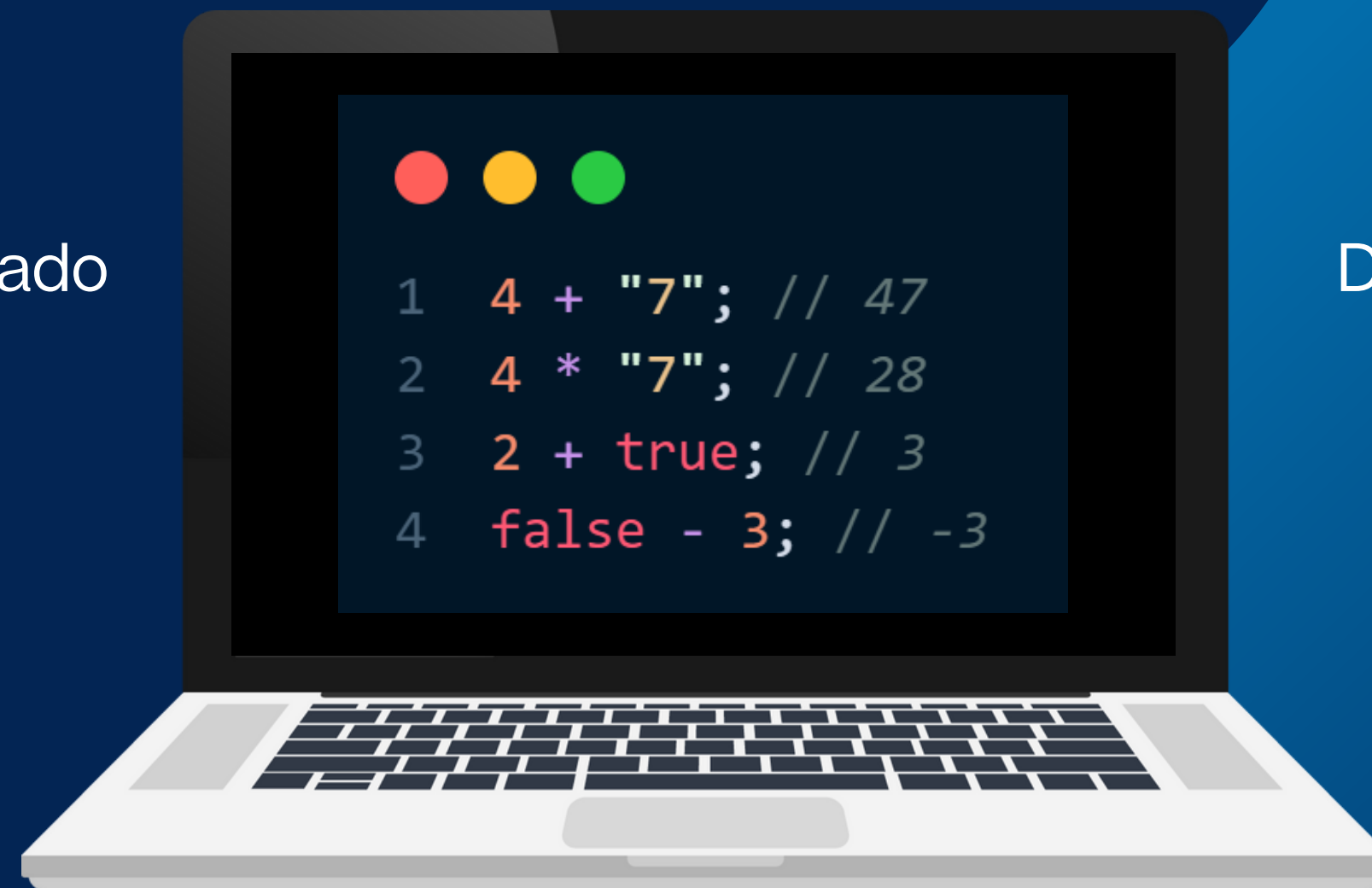
¿QUÉ TIPO DE LENGUAJE ES JAVASCRIPT?

Lenguaje Interpretado

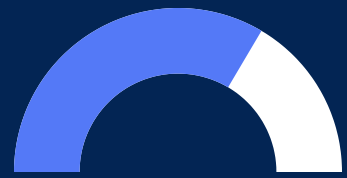
Orientado a Objetos

Débilmente Tipado

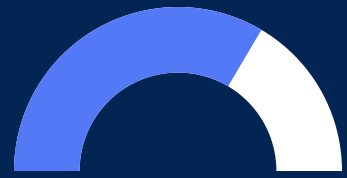
Dinámico



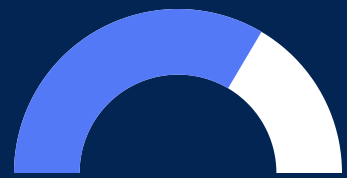
¿PARA QUÉ PUEDO USAR JAVASCRIPT?



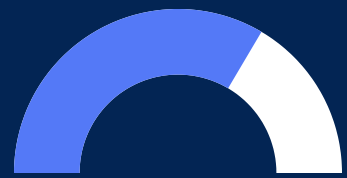
Añadir **características interactivas** a páginas o aplicaciones web



Generar contenido de **forma dinámica**



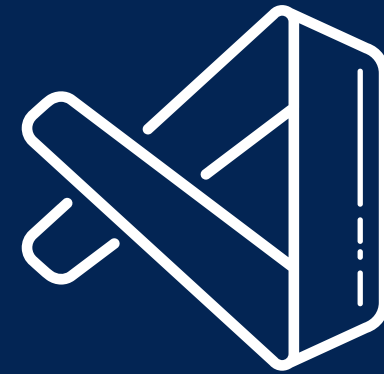
Interfaces de Programación de Aplicaciones del Navegador



Construir librerías, apps móviles y desktop, machine learning

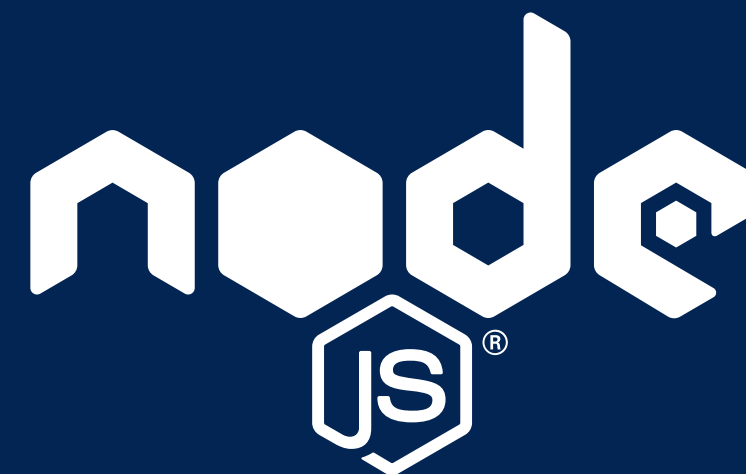


APLICACIONES CONSTRUIDAS CON JAVASCRIPT



¿CÓMO EJECUTAR ARCHIVOS DE JAVASCRIPT?

Ejecutar archivos en consola

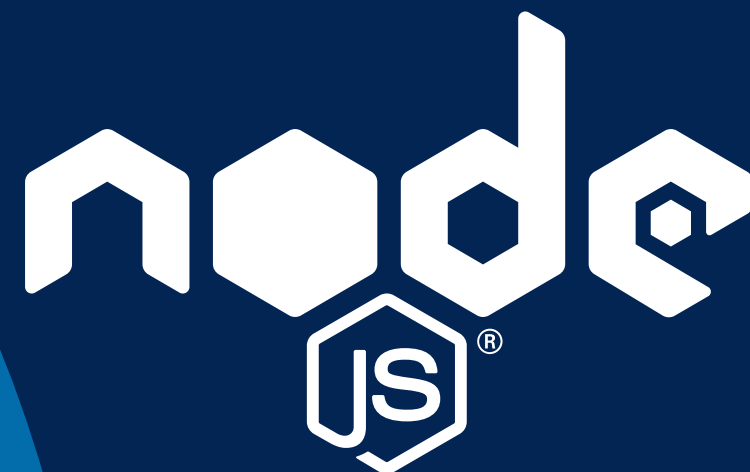


Chrome dev tools



Integración de HTML, CSS, JS





```
1 console.log('hello world');
```

node nombre_archivo.js



```
1 var text = document.createElement("h1");  
2 text.innerHTML = "Estamos creando un nuevo archivo en Google :)";  
3 document.body.prepend(text);
```

Inspect - Sources - Snippets - New Snippet - Ctrl+Enter



```
1 <script src="functions.js"></script>
```



```
1 var text = document.createElement('h1');  
2 text.innerHTML = "Este será nuestro título";  
3 document.body.prepend(text);  
4 console.Log('Generando contenido en consola');  
5 console.Log('Testing my app: 2+2= '+(2+2));
```



VARIABLES EN JAVASCRIPT

Las variables son contenedores para almacenar información. Usa nombres cortos para sus identificadores. Deben comenzar con una letra, pueden contener guion bajo, números, \$, letras. Palabras reservadas.

VAR

Funcionamiento anticuado, tiene un diseño confuso para la industria actual. Aplica hoisting

LET

No soporta hoisting. Se recomienda usar en lugar de var. Almacena datos que pueden cambiar a lo largo del programa.

CONST

Usada para valores que no cambian en la ejecución del programa



TIPOS DE DATOS EN JAVASCRIPT



Boolean

Solo permite valores dos tipos de valores que son, True o False.



Null

Es una palabra especial que indica que el dato arroja un valor null.

Undefined

Es una propiedad de alto nivel cuyo valor no ha sido definido, es decir, pudo haber sido declarado pero no se le ha definido o asignado algún valor.

Number

Es un dato de tipo número que permite números enteros y decimales.

String

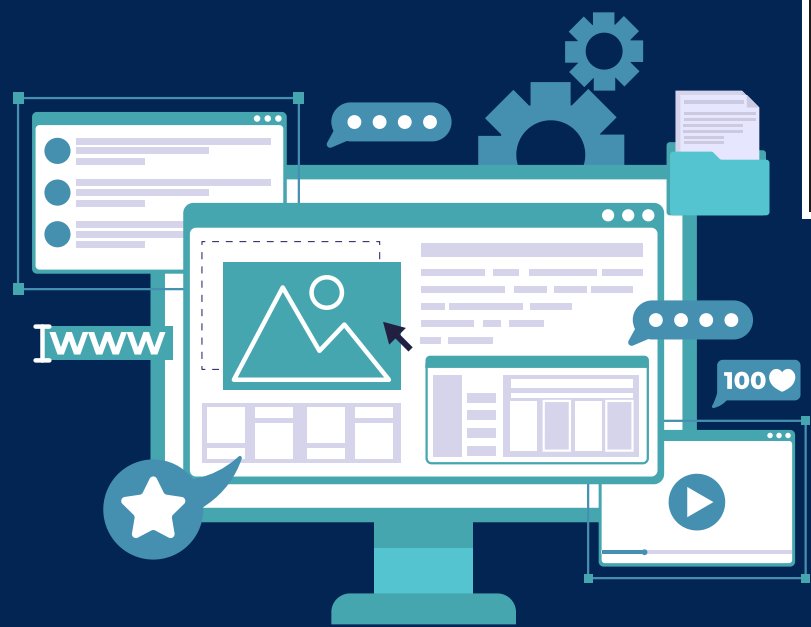
Indica que el contenido es una cadena de caracteres que representa un texto

Object

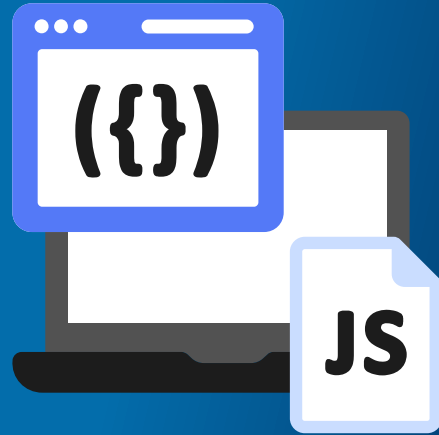
Es un tipo de dato que contiene datos e instrucciones para trabajar con los datos.

ASIGNACIONES CON OPERADORES

Nombre	Operador abreviado	Significado
Asignación	$x = y$	$x = y$
Asignación de adición	$x += y$	$x = x + y$
Asignación de resta	$x -= y$	$x = x - y$
Asignación de multiplicación	$x *= y$	$x = x * y$
Asignación de división	$x /= y$	$x = x / y$
Asignación de residuo	$x \% = y$	$x = x \% y$
Asignación de exponenciación	$x ** = y$	$x = x ** y$



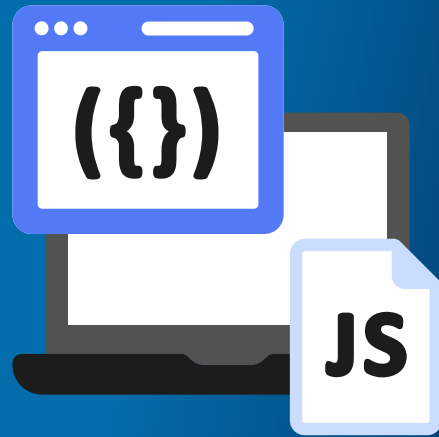
ANATOMIA DE UNA FUNCION EN JAVASCRIPT



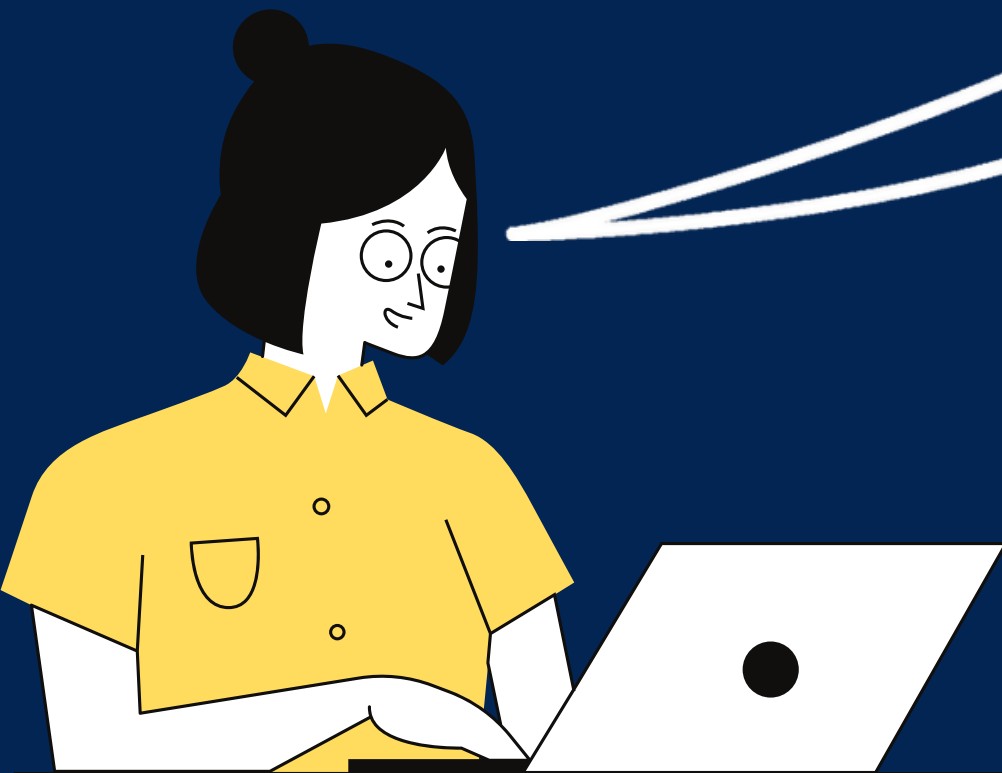
Una función es una secuencia de instrucciones de código en un programa.

En JS se define:

```
1  function nombre_func(param1, param2, param3) {  
2      // statements - body function  
3      console.log('Esta es mi primera funcion');  
4      let suma = param1+param2+param3;  
5      return suma;  
6  }  
7  
8  let resultado = nombre_func(2,3,10);  
9  
10 console.log(resultado);
```



Pero hoy día existen maneras
más cómodas de escribir
funciones, veamos las famosas
arrow functions

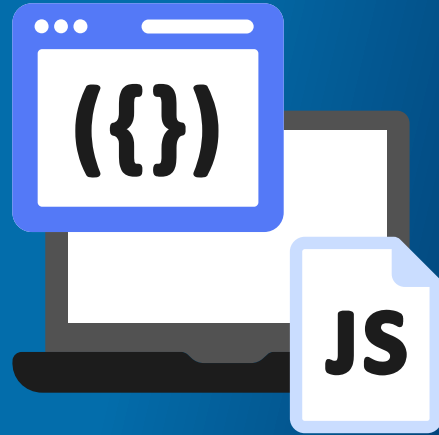


```
const greet = (who) => {  
  return `Hello, ${who}!`;  
}
```

VS

```
const greet = function(who) {  
  return `Hello, ${who}`;  
}
```

ARROW FUNCTION



Son una forma alternativa de declarar funciones en JS, y se caracteriza por ser compacta y precisa. Es la forma en la que los frameworks como React hacen sus declaraciones

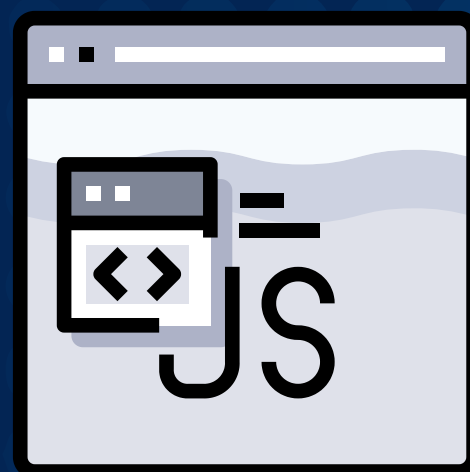
```
const add2 = (a) => a + 2
```

Regular function:

```
function add2(a) {  
  return a + 2  
}
```



PRACTICAS



PONGAMOS ESE CEREBRO A TRABAJAR!

Conviertan la siguiente función a una **arrow function**.



```
1  function nombre_func(param1, param2, param3) {  
2      // statements - body function  
3      console.log('Esta es mi primera funcion');  
4      let suma = param1+param2+param3;  
5      return suma;  
6  }  
7  
8  let resultado = nombre_func(2,3,10);  
9  
10 console.log(resultado);
```

HABLEMOS SOBRE EL SCOPE DE VARIABLES

El scope de una variable hace referencia al lugar donde podrá ser accesible, en JavaScript tenemos varias opciones: global, local y bloque.



HABLEMOS SOBRE EL SCOPE DE VARIABLES



GLOBAL SCOPE

Las variables globales son accesibles en cualquier lugar de nuestro código. Esto sucede porque cuando creamos una variable global esta, es instanciada en el objeto window.

```
1 let variable = 'esto es una variable';
2
3 function print() {
4   console.log(variable+' llamada dentro de una funcion');
5 };
6
7 print();
8 console.log(variable+' llamada fuera de una funcion');
```

HABLEMOS SOBRE EL SCOPE DE VARIABLES

LOCAL SCOPE

Las variables solo van a vivir en la función donde son creadas, para esto utilizamos

```
1 function variables() {  
2     let variableLocal = 'Esto es una variable local';  
3     console.log(variableLocal+' llamada desde mi funcion');  
4 }  
5  
6 variables();  
7  
8 console.log(variableLocal+' llamada desde fuera de la funcion');
```



HABLEMOS SOBRE EL SCOPE DE VARIABLES

BLOCK SCOPE

Desde 2015 con la llegada de `let` y `const`, se puede aplicar scope dentro de bloques de código. Un bloque de código son todas las instrucciones que se encuentran dentro de llaves `**{codigo}**`

```
1 function block() {  
2     const count = 5;  
3     for(let i = 0; i < count; i++) {  
4         console.log('Numero: ' + i);  
5     }  
6     console.log(i);  
7 }  
8 block();
```



CONDICIONALES

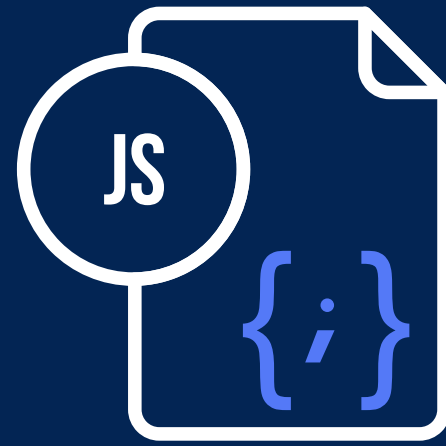


```
const mayorEdad = 18;  
if(mayorEdad >= 18) {  
    console.log("Es mayor de edad");  
}else {  
    console.log("Es menor de edad");  
}
```

OPERADORES LOGICOS EN JAVASCRIPT

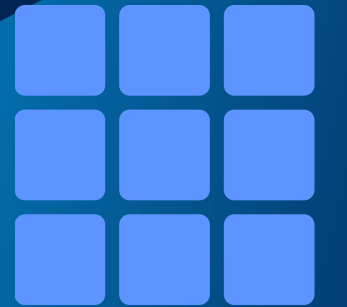
```
let var1=3;  
let var2=4;
```

Operador	Descripción	Ejemplos que devuelven true
Igual (==)	Devuelve true si los operandos son iguales.	3 == var1 "3" == var1 3 == '3'
No es igual (!=)	Devuelve true si los operandos no son iguales.	var1 != 4 var2 != "3"
Estrictamente igual (===)	Devuelve true si los operandos son iguales y del mismo tipo.	3 === var1
Desigualdad estricta (!==)	Devuelve true si los operandos son del mismo tipo pero no iguales, o son de diferente tipo.	var1 !== "3" 3 !== '3'
Mayor que (>)	Devuelve true si el operando izquierdo es mayor que el operando derecho.	var2 > var1 "12" > 2
Mayor o igual que (>=)	Devuelve true si el operando izquierdo es mayor o igual que el operando derecho.	var2 >= var1 var1 >= 3
Menor que (<)	Devuelve true si el operando izquierdo es menor que el operando derecho.	var1 < var2 "2" < 12
Menor o igual que (<=)	Devuelve true si el operando izquierdo es menor o igual que el operando derecho.	var1 <= var2 var2 <= 5



ARRAYS EN JAVASCRIPT

Son una colección de elementos almacenados en memoria e identificados por una misma variable, donde cada elemento puede ser accedido mediante un índice único.



```
let pais = ['Mexico', 'España', 'Argentina', 'Chile', 'Colombia', 'Venezuela', 'Perú', 'Costa Rica'];
```

```
let datos = ['Frío', 33, false, 'nube', -11.22, true, 3.33, 'variado'];
```

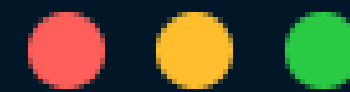
```
let fruta = [];
```

```
let fruta = new Array();
```



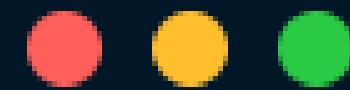
OPERACIONES CON ARRAYS

Acceder a un elemento específico del array



```
1 const element = array[index];
```

Conocer la longitud del array



```
1 const longitud = array.length;
```

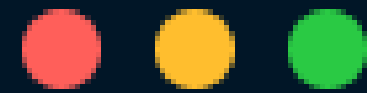
Agregar un elemento al final del array



```
1 array_name.push('objeto');
```

OPERACIONES CON ARRAYS

Agregar un elemento al inicio del array



```
1 array_name.unshift('objeto');
```

Imprimir todos los elemntos del array junto con su índice




```
1 for(let i=0; i<array.length; i++) {  
2     console.log('Elemento en indice '+i+ ' es '+array[i]);  
3 }
```

OPERACIONES CON ARRAYS

Cada vez que llamas el método `pop()`, este elimina un elemento del final de un arreglo.



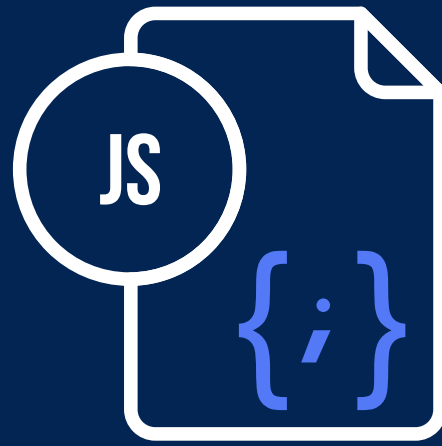
```
1 array_name.pop();
```



```
1 array_name.shift();
```

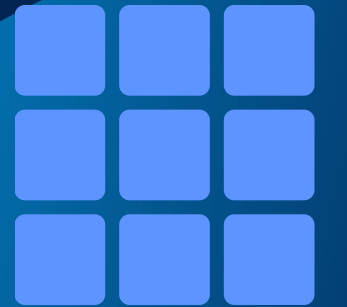
Usa el método `shift()` para eliminar un elemento desde el principio del arreglo.

Como el método `pop()` y `shift()` se retorna el elemento eliminado y cambia el arreglo original.



OBJETOS EN JAVASCRIPT

En JavaScript, un objeto es una entidad independiente con propiedades y tipos. Compáralo con una taza, por ejemplo. Una taza es un objeto con propiedades. Una taza tiene un color, un diseño, un peso, un material del que está hecha, etc.



Lo importante es que todos se pueden declarar usando **JSON**



```
var obj = { property_1: value_1, // property_# puede ser un identificador...  
            2:          value_2, // o un número...  
            // ...,  
            'property n': value_n }; // o una cadena
```



ASINCRONISMO EN JAVASCRIPT



JavaScript es un lenguaje de programación asincrono. Lo que quiere decir esto es que al ejecutar código JavaScript el hilo de ejecución continuará a pesar de encontrarse en situaciones en las que no obtenga un resultado inmediatamente.

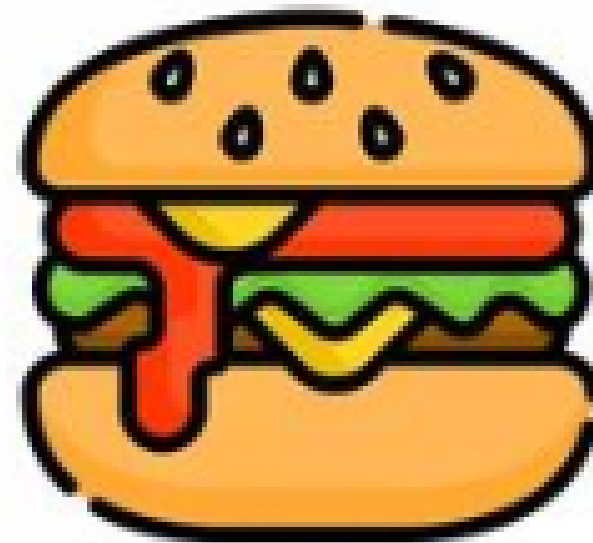
Esto algunas veces es un problema, pero JS ofrece alternativas de solución para esto, como lo son los callbacks, para visibilizar la asincronia, y las promesas, que vuelven el código sincrónico.

¿Cómo podemos
explicar esto de una
manera más natural?

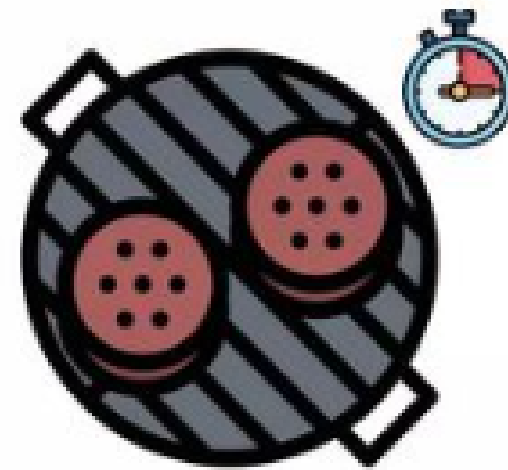


Pasos:

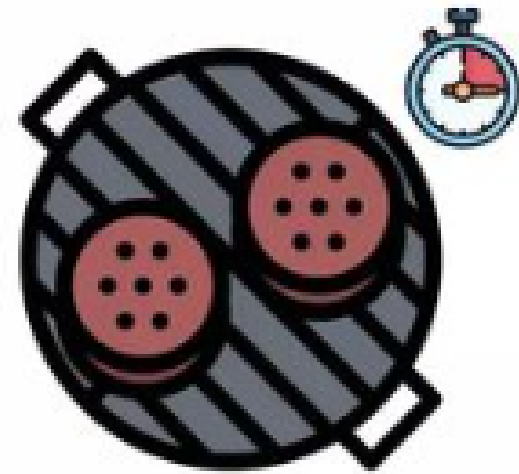
- 1 Partir el pan a la mitad.
- 2 Untar aderezo en los panes.
- 3 Cocer la carne.
- 4 Cortar jitomate y lechuga.
- 5 Agregar queso.



Cuando ponemos a cocer la carne,
la dejamos sobre la plancha y
esperamos un tiempo mientras
el fuego le quita lo crudo.



Cocer la carne es un **proceso asíncrono** porque, mientras el resto de pasos se sigue realizando, este en particular **toma un tiempo en completarse.**





```
function exitoCallback(resultado) {  
    console.log("Archivo de audio disponible en la URL " + resultado);  
}  
  
function falloCallback(error) {  
    console.log("Error generando archivo de audio " + error);  
}  
  
crearArchivoAudioAsync(audioConfig, exitoCallback, falloCallback);  
  
//La forma corta seria =  
const promesa = crearArchivoAudioAsync(audioConfig);  
promesa.then(exitoCallback, falloCallback);
```

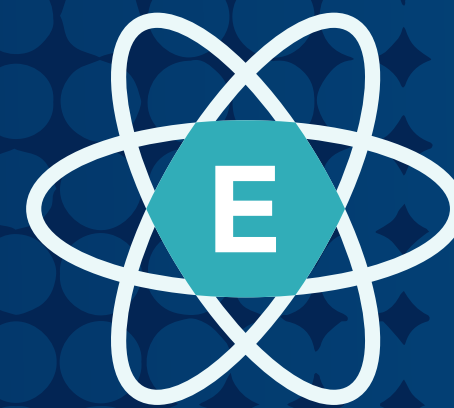


```
async function foo() {  
  await 1;  
}
```

```
const getServicioByCod = async (req,res) => {  
  const codlineaux = req.params.codlinea;  
  const response = await pool.query('SELECT * FROM SERVICIO WHERE cod_servicio = $1', [codlineaux]);  
  res.json(response.rows);  
};
```

INTRODUCCION A FRAMEWORKS

Un Framework es una estructura previa que se puede aprovechar para desarrollar un proyecto, simplifica la elaboración de una tarea, ya que solo es necesario complementarlo de acuerdo a lo que se quiere realizar.





@ceinf_ucabg



@cde_ucabg



¡Nunca pares de aprender!

**Agradecimientos a
Paulina Espejo por el Material
Educativo ♥**