DEV.F::

Clase 1

Introducción a ECMAScript

Comenzamos en 10 min

6:40pm (hora CDMX)

Temas de la clase (180 min)

¿Qué es ECMAScript?

Historia
Evolución

Funciones

Var, Let, Const
Arrow Functions
String Literal

Conceptos Avanzados

Asincronismo
Promesas

Reto de la semana

DEV.F::

¿Qué es ECMAScript?

ECMAScript es el estándar y no el lenguaje. Piensa en él como el plano que describe cómo debe funcionar JavaScript.

Puede llegar a tomar mucho tiempo su implementación en un navegador.

Ecma International está a cargo de estandarizar este lenguaje de programación, a través de una serie de versiones que **añaden funcionalidades nuevas.**

https://ecma-international.org/

European Computer Manufacturers Association

- ECMA International es una organización sin fines de lucro que se dedica a la estandarización de las tecnologías de la información y la comunicación.

  ¿Qué significa esto?

- Básicamente, se aseguran de que diferentes dispositivos y software puedan "hablar" entre sí sin problemas.

Evolución de ECMAScript

Inicios (1995-1999):

- **1995:** Nace JavaScript (originalmente llamado Mocha y luego LiveScript) de la mano de Brendan Eich en Netscape.
- **1997:** Se crea el estándar ECMAScript 1 (ES1) basado en JavaScript, buscando la unificación y la interoperabilidad entre navegadores.
- **1998:** ECMAScript 2 (ES2) se publica con pequeñas actualizaciones para alinear la especificación con el estándar ISO/IEC 16262.
- **1999:** ECMAScript 3 (ES3) introduce mejoras significativas como expresiones regulares, manejo de excepciones (`try-catch`) y mejor soporte para texto. Esta versión fue ampliamente adoptada y sentó las bases para el JavaScript que conocemos hoy.

Evolución de ECMAScript

Un período de estancamiento (2000-2008):

- 2000 - 2008: El desarrollo de ECMAScript 4 (ES4) se vuelve complejo y controversial, con desacuerdos sobre la dirección del lenguaje. Finalmente, ES4 es abandonado.

Renacimiento y evolución rápida (2009-presente):

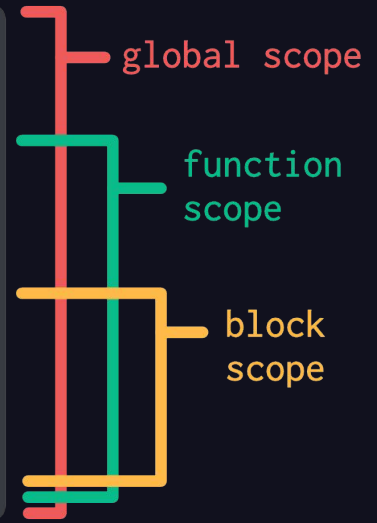
- 2009: Se publica ECMAScript 5 (ES5) con mejoras incrementales en ES3, incluyendo nuevas funciones para arrays, manejo de objetos y "strict mode" para un código más robusto.
- 2015: ¡Un gran salto! ECMAScript 6 (ES6 o ES2015) trae una gran cantidad de nuevas características: funciones flecha, clases, template literals, promesas, módulos, etc. Esta versión moderniza JavaScript y lo hace más potente y expresivo.
- Desde 2016: ECMA International adopta un ciclo de lanzamientos anual. Cada año se publica una nueva versión de ECMAScript con mejoras incrementales. ES2016, ES2017, ES2018, ES2019, ES2020, ES2021, etc. han ido añadiendo características como async/await, operadores de propagación, optional chaining, etc.

Declaración de variables con let y const

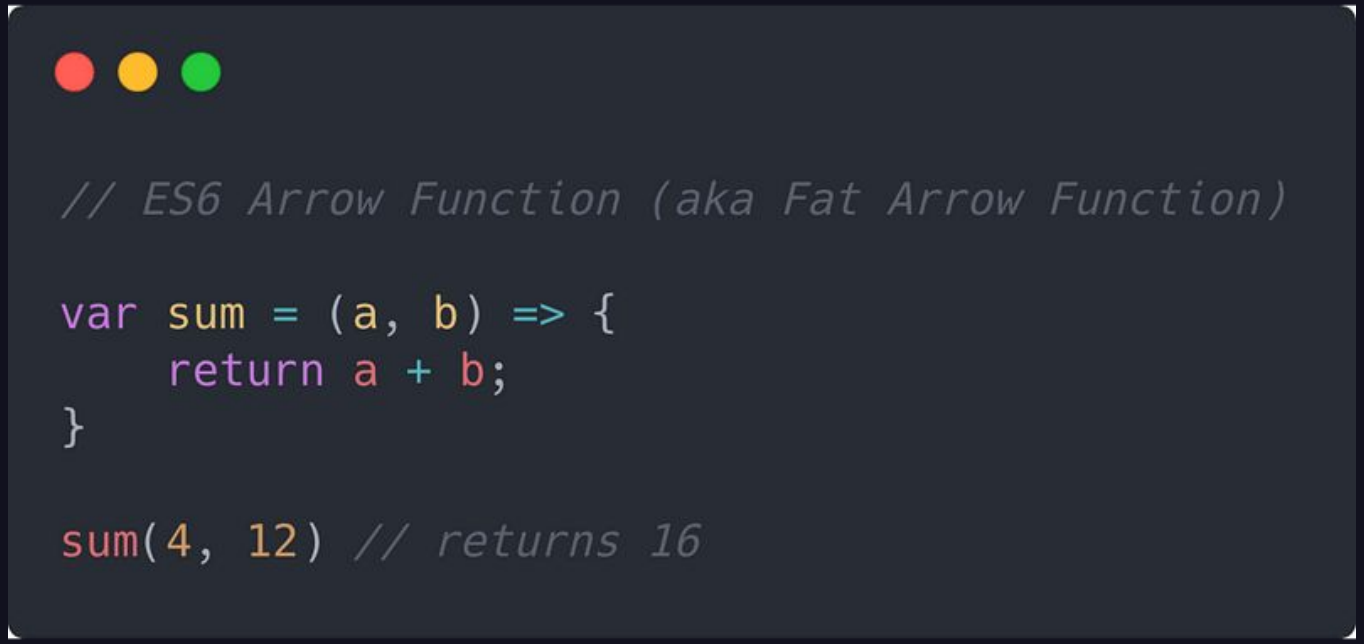
VAR vs LET vs CONST

	var	let	const
Reasignación	✓	✓	✗
Redeclaración	✓	✗	✗
Propiedad del objeto global (`window`)	✓	✗	✗
Function Scope	✓	✓	✓
Block Scope	✗	✓	✓
Hoisting	✓	✓	✓
TDZ (Temporal Dead Zone)	✗	✓	✓

```
// Global Scope
var var1 = 1;
let let1 = 1;

function myFunction(){
  // Function Scope
  var var2 = 2;
  let let2 = 2;

  for(var i = 0; i < 1; i++){
    // Block Scope
    var var3 = 3;
    let let3 = 3;

  }
}
```



Arrow functions (funciones de flecha)



```
// ES6 Arrow Function (aka Fat Arrow Function)

var sum = (a, b) => {
    return a + b;
}

sum(4, 12) // returns 16
```

Partes de un Arrow Function

Parámetros: Los parámetros se colocan entre paréntesis ( ), de manera similar a como se hacen en las funciones tradicionales.

Símbolo de flecha =>: Este símbolo indica que estás declarando una función de flecha.

Cuerpo de la función: Puede tener dos formas:

Si el cuerpo de la función es una sola expresión, puedes escribirlo después de la flecha sin necesidad de utilizar { } y la función devolverá automáticamente el valor de esa expresión.

Si el cuerpo de la función requiere múltiples líneas o más lógica, entonces debes utilizar { } para definir un bloque de código y utilizar la palabra clave return explícitamente si deseas devolver un valor.

DEV.F::

Template literals para crear cadenas de texto interpoladas de manera más legible.

```
`string ${literal}`
```

Template literal

La sintaxis básica de un template literal consiste en encerrar la cadena de texto entre comillas invertidas ( ) en lugar de comillas simples (' ') o dobles (" "), lo que permite incluir fácilmente expresiones de JavaScript dentro de la cadena utilizando ${ }. Esto significa que puedes insertar valores de variables, resultados de funciones o incluso otras expresiones dentro de la cadena de manera más concisa y legible.

DEV.F::

Symbol, un nuevo tipo de dato primitivo para crear identificadores únicos

```
// Every symbol created with Symbol() is unique.
console.log(Symbol() === Symbol()) // false
console.log(Symbol("✨") === Symbol("✨")) // false

// Calling Symbol.for() makes a global symbol.
console.log(Symbol.for("✨") === Symbol.for("✨")) // true

// You can check for a symbol using typeof.
console.log(typeof Symbol()) // "symbol"
```

Symbol

En JavaScript, Symbol es un tipo de dato primitivo introducido en ECMAScript 6 (también conocido como ES6). Un Symbol es un **valor único e inmutable** que se puede utilizar como clave para las propiedades de los objetos. Cada valor de Symbol es único y **no se puede replicar ni igualar a otro valor de Symbol,** lo que lo hace útil para crear identificadores únicos.

Los Symbol se crean utilizando la función constructora Symbol() sin la palabra clave new. No se pueden crear con la palabra clave new porque son primitivos y no objetos. Además, los Symbol pueden tener una descripción opcional que proporciona información descriptiva sobre el Symbol, pero esta descripción no afecta a su unicidad.

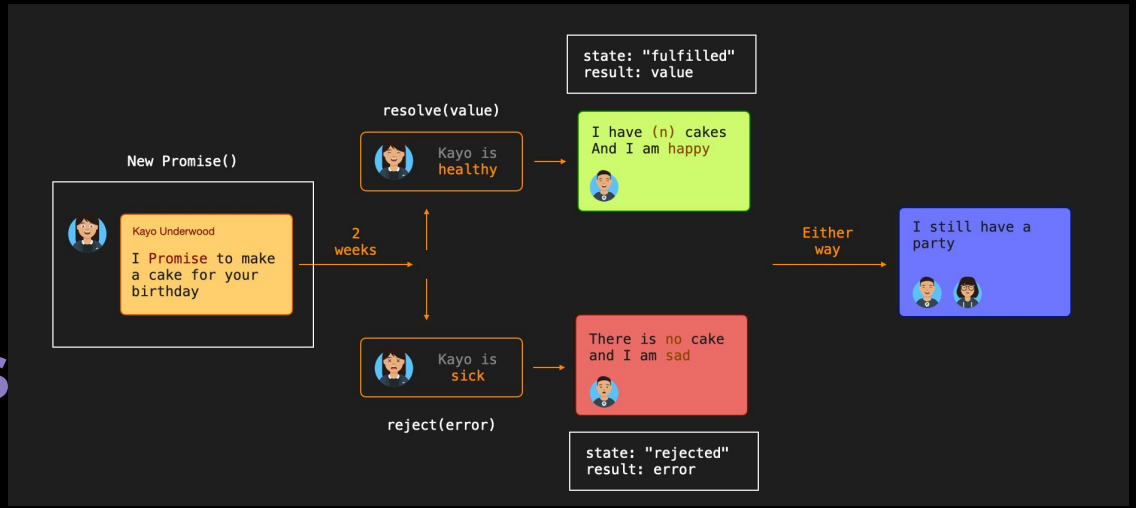
Desestructuración de objetos y arreglos

```
let options = {
  title: "Menu",
  width: 100,
  height: 200
};

// Solo extraer el título como una variable
let { title } = options;

console.log(title); // Menu
```

Módulos para importar y exportar código entre archivos.

```
import {Perro, Oso} from './Animal';
export class Zoologico {
    constructor() {
      const perro = new Perro();
      const oso = new Oso()p;
    }
}
```

Promesas para trabajar con operaciones asincrónicas de manera más estructurada.



Funciones asincrónicas (async/await) para escribir código asincrónico de forma más legible y estructurada

Ejercicio: Generador de Tarjetas de Presentación

Objetivo: Crear un programa que genere tarjetas de presentación personalizadas utilizando funciones de flecha y template literals.

Instrucciones:

1. **Crear un arreglo de objetos:** Define un arreglo llamado contactos que contenga al menos 3 objetos, cada uno representando la información de una persona (nombre, puesto, empresa, correo electrónico, número de teléfono).
2. **Función de flecha para generar la tarjeta:** Escribe una función de flecha llamada generarTarjeta que reciba un objeto contacto como parámetro y utilice template literals para construir una cadena de texto que represente la tarjeta de presentación. La tarjeta debe incluir el nombre, puesto, empresa, correo electrónico y número de teléfono del contacto, formateados de manera clara y legible.
3. **Mostrar las tarjetas:** Utiliza un bucle para recorrer el arreglo contactos y, para cada contacto, llama a la función generarTarjeta y muestra la tarjeta resultante en la consola.


No olviden
Las lecturas de
edu.devf.la
ESCUELA DE HACKERS
DEVF FUNCTION ()
CONST : DEV.F
GENIO O HACKER?
404
DEV.F::