ES6

¿Qué es?

ES6 (también conocido como ECMAScript 2015) es la sexta versión del estándar de JavaScript. Fue aprobado por la organización ECMA International en 2015 y fue una actualización importante del lenguaje que introdujo muchas características nuevas y mejoras a JavaScript.

Algunas de las características más notables de ES6 incluyen:

- Arrow functions: una forma más concisa de escribir funciones en JavaScript.
- Let y const: nuevas formas de declarar variables que reemplazan la palabra clave "var".
- Plantillas de cadenas: una forma más fácil de concatenar cadenas de texto en JavaScript.
- Desestructuración: una forma de extraer valores de objetos y arreglos en variables separadas.
- Clases: una forma más clara y orientada a objetos de crear objetos en JavaScript.
- Promesas: una forma de manejar operaciones asincrónicas en JavaScript.

En general, ES6 hizo que JavaScript fuera más fácil de leer, escribir y mantener. Desde su lanzamiento, se han lanzado varias versiones posteriores del estándar ECMAScript, cada una con nuevas características y mejoras.

Ejemplos de las características:

- Variables: let y const
    - Las variables let y const se introdujeron en ES6 para reemplazar la antigua palabra clave var para declarar variables.
    - let se usa para declarar variables que pueden cambiar su valor, mientras que const se usa para declarar variables cuyo valor no puede ser cambiado después de la asignación inicial.

  Ejemplo:

```
let a = 10;
const b = 20;
```

- **Arrow functions**
  Las funciones de flecha o arrow functions son una forma más concisa de escribir funciones en JavaScript.
  Ejemplo:

```
// Función tradicional
function sum(a, b) {
  return a + b;
}

// Arrow function
const sum = (a, b) => a + b;
```

- **Template literals**
  Las plantillas de cadenas o template literals son una forma más fácil de concatenar cadenas de texto en JavaScript.
  Ejemplo:

```
const name = "Juan";
console.log(`Hola ${name}, ¿cómo estás?`);
```

- **Desestructuración**
  La desestructuración es una forma de extraer valores de objetos y arreglos en variables separadas.
  Ejemplo:

```
const person = {
  name: "Juan",
  age: 30,
  country: "Mexico"
};

const { name, age } = person;
console.log(name); // "Juan"
console.log(age); // 30
```

- **Clases**
  Las clases son una forma más clara y orientada a objetos de crear objetos en JavaScript.
  Ejemplo:

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  sayHello() {
    console.log(`Hola, mi nombre es ${this.name} y tengo ${this.age} años.`);
  }
}

const person = new Person("Juan", 30);
person.sayHello(); // "Hola, mi nombre es Juan y tengo 30 años."
```

- **Promesas**
  Las promesas son una forma de manejar operaciones asincrónicas en JavaScript.
  Ejemplo:

```
function fetchData() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve("Datos recuperados correctamente.");
    }, 2000);
  });
}

fetchData()
  .then(response => console.log(response))
  .catch(error => console.error(error));
```

- **Spread operator**
  El operador de propagación o spread operator es una forma de descomponer un objeto o un arreglo en sus elementos individuales.
  Ejemplo:

```
const numbers = [1, 2, 3, 4];
console.log(...numbers); // 1 2 3 4

const person = {
  name: "Juan",
  age: 30,
  country: "Mexico"
};

console.log({...person}); // { name: "Juan", age: 30, country: "Mexico" }
```

- **Default parameters**
  Los parámetros por defecto o default parameters son una forma de proporcionar valores predeterminados para los parámetros de una función.
  Ejemplo:

```
function greet(name = "amigo") {
  console.log(`Hola, ${name}!`);
}

greet(); // "Hola, amigo!"
greet("Juan"); // "Hola, Juan!"
```

Aplicando buenas practicas

- Nombres de variables y funciones significativos: utiliza nombres descriptivos para tus variables y funciones, que indiquen claramente su propósito y lo que hacen.

- Evita las variables globales: en lugar de utilizar variables globales que puedan ser accedidas y modificadas desde cualquier parte del código, utiliza variables locales que sean visibles solo en la función en la que se definen.

- Usa comentarios para documentar tu código: los comentarios pueden ayudar a explicar el propósito de tu código y cómo funciona, especialmente para aquellos que lo leerán en el futuro.

- Usa un formato consistente: utiliza un formato consistente para tu código, como la indentación y el uso de espacios y saltos de línea, para que sea más fácil de leer y mantener.

- Evita la repetición de código: utiliza funciones o módulos para encapsular el código que se utiliza en varias partes de tu programa, en lugar de copiar y pegar el mismo código en múltiples lugares.

- Utiliza pruebas unitarias: escribe pruebas automatizadas para probar tu código, para asegurarte de que funciona correctamente y para detectar errores antes de que sean implementados.

- Sigue los principios SOLID: los principios SOLID son una guía para escribir código orientado a objetos que sea fácil de mantener y extender. Estos principios incluyen la responsabilidad única, la abierta/cerrada, la sustitución de Liskov, la segregación de interfaces y la inversión de dependencias.

- Utiliza herramientas de control de versiones: utiliza herramientas de control de versiones como Git para realizar un seguimiento de los cambios en tu código, para que puedas revertir cambios si es necesario y colaborar con otros programadores en el mismo código.

- Aprende de las mejores prácticas de otros programadores: lee el código de otros programadores y aprende de sus mejores prácticas, para que puedas mejorar tu propio código.

Estas son solo algunas de las muchas buenas prácticas que puedes seguir como programador. Lo más importante es ser constante y estar dispuesto a aprender y mejorar continuamente tu código.

```
// Esta función calcula el área de un círculo dado su radio
function calcularAreaCirculo(radio) {
  // Si el radio no es un número o es menor o igual a cero, se lanza un error
  if (typeof radio !== "number" || radio <= 0) {
    throw new Error("El radio debe ser un número mayor que cero");
  }

  // Se calcula el área del círculo utilizando la fórmula A = pi * r^2
  const area = Math.PI * Math.pow(radio, 2);

  // Se devuelve el área redondeada a dos decimales
  return area.toFixed(2);
}

// Ejemplo de uso
const radio = 5;
const area = calcularAreaCirculo(radio);
console.log(`El área de un círculo de radio ${radio} es ${area}`);
```

En este código, podemos ver algunas buenas prácticas:

- Se utiliza un nombre de función significativo (calcularAreaCirculo) que indica claramente lo que hace la función.
- Se realiza una validación del parámetro de entrada para asegurarse de que sea un número mayor que cero.
- Se utiliza una constante para almacenar el valor de pi, que se utiliza en el cálculo del área.

- Se utiliza el método Math.pow() para elevar el radio al cuadrado, en lugar de utilizar el operador ** para mejorar la legibilidad del código.
- Se utiliza el método toFixed() para redondear el resultado del cálculo del área a dos decimales.
- Se utiliza un literal de plantilla para imprimir un mensaje con el valor del radio y del área.

Además, se han incluido comentarios en el código para explicar lo que hace cada parte del código y por qué se han tomado ciertas decisiones.

Ejercicios prácticos

1. Crear una función que tome un array de números y devuelva un array con solo los números pares.
2. Crear una función que tome un objeto y devuelva un array de sus valores.
3. Crear una función que tome un array de strings y devuelva un objeto que tenga como claves cada una de las palabras y como valor la cantidad de veces que aparece esa palabra en el array.
4. Crear una función que tome un array de objetos y devuelva un array con solo los objetos que tengan una propiedad específica.
5. Crear una función que tome un array de strings y devuelva un array con las mismas palabras, pero en orden alfabético.
6. Crear una función que tome dos objetos y devuelva un objeto con las propiedades de ambos objetos.
7. Crear una función que tome un array de números y devuelva la suma de todos los números.
8. Crear una función que tome un array de números y devuelva un array con los mismos números, pero multiplicados por un valor dado.
9. Crear una función que tome una cadena de texto y devuelva la misma cadena en orden inverso.
10. Crear una función que tome un objeto y devuelva un array de sus claves.