Fundamentos de programación

0.7 Funciones

0.7.2 Parámetros por valor y por referencia

¿Qué es un Objeto?
En Javascript, un objeto es un contenedor que puede almacenar variables y funciones internas. Estas variables y funciones se denominan propiedades y métodos, respectivamente.

Ejemplo de Declaración de un Objeto

```
let producto = {
    nombre: "Monitor de 20 pulgadas",
    precio: 300
};
```

En este ejemplo, **producto** es un objeto con dos propiedades: **nombre** y **precio**.
Creación y Uso de Arrays de Objetos
Declaración de un Array de Objetos

```
let carrito = [
    { nombre: "Monitor de 20 pulgadas", precio: 300 },
    { nombre: "Teclado mecánico", precio: 100 },
    { nombre: "Mouse", precio: 50 }
];
```

Aquí, **carrito** es un array que contiene tres objetos, cada uno representando un producto con sus respectivas propiedades.

Acceso a los Elementos del Array

Los elementos del array se pueden acceder utilizando sus índices:

```
console.log(carrito[0].nombre); // "Monitor de 20 pulgadas"
```

Cálculo del Total de Precios en un Array

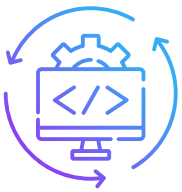
Uso de un Bucle para Calcular el Total

```
let total = 0;
for (let i = 0; i < carrito.length; i++) {
    total += carrito[i].precio;
}
console.log("Total a pagar:", total);
```

En este bucle, se itera sobre el array **carrito**, sumando el precio de cada producto a la variable **total**.

Simplificación del Código con Funciones

Extracción de la Lógica en una Función
Para hacer el código más legible y modular, podemos extraer la lógica de cálculo a una función:




```
function calcularTotal(carrito) {
    let total = 0;
    for (let i = 0; i < carrito.length; i++) {
        total += carrito[i].precio;
    }
    return total;
}

let totalAPagar = calcularTotal(carrito);
console.log("Total a pagar:", totalAPagar);
```

Esta función, **calcularTotal**, toma un array de productos como argumento y devuelve el total de sus precios.

Mejores Prácticas y Recomendaciones

Uso de Nombres Significativos
Es importante dar nombres significativos a las funciones para mejorar la legibilidad del código. Por ejemplo, **calcularTotal** es más descriptivo y fácil de entender que un simple bucle **for**.

Modularización del Código
Separar la lógica en funciones ayuda a mantener el código limpio y manejable. Esto facilita la identificación y solución de problemas.