¡Hola amiguitos del código! 🌟 Hoy vamos a hablar sobre un concepto muy importante en JavaScript: las funciones.

¡Sí, esas cosas mágicas que hacen que nuestro código haga cosas increíbles!

# Planteamiento del problema

// Calcular media de 3 números e irlos mostrando por pantalla

// Imaginemos que los va introduciendo un usuario

// Valores 3,5,6

// Valores 1,8,5

// Valores 2,3,5

// Valores 3.3, 98, 32

// Primera instancia

// Declaración/inicialización de variables, cálculo y muestra de resultados

let num1 = 3

let num2 = 5

let num3 = 6

let promedio = num1 + num2 + num3 / 3

console.log(promedio)

// Segunda instancia

num1 = 1

num2 = 8

num3 = 5

promedio = num1 + num2 + num3 / 3

console.log(promedio)

// Tercera instancia

num1 = 2

num2 = 3

num3 = 5

promedio = num1 + num2 + num3 / 3

console.log(promedio)

// Cuarta instancia

num1 = 3.3

num2 = 98

num3 = 32

promedio = num1 + num2 + num3 / 3

console.log(promedio)

# Declaración de Función 📜

function calcularPromedio () {

  let num1 = 3

  let num2 = 5

  let num3 = 6

  let promedio

  // Pasos para calcular y mostrar el promedio aquí

}

En JavaScript, eso sería una declaración de función. Decimos: "Hey, aquí hay una operación de cálculo llamada `calcularPromedio`. Cuando la llamemos, ¡hace magia y hace el promedio!"

# Parámetros 🎁

Ahora, ¿y si queremos que nuestro cálculo sean de diferentes números? ¡Necesitamos decirle a nuestra función qué números intervienen en la operación! Ahí es donde entran los "parámetros".

function calcularPromedio (num1, num2, num3) {

  let promedio

  // Pasos para calcular y mostrar el promedio aquí

}

Ahora, cuando llamamos `calcularPromedio(num1, num2, num3)` y pasamos los argumentos 1, 2 y 3 ¡obtenemos 2! 🍫

# Valor de Retorno 🎁➡️

¡Ahora viene lo divertido! Después de calcular el promedio, ¿queremos mostrar siempre el resultado en la función o almacenar su valor para su uso posterior? ¡Eso es lo que llamamos "valor de retorno"!

function calcularPromedio (num1, num2, num3) {

  // Pasos para calcular el promedio y retornar su valor

  return // Retornar el valor final del promedio aquí

}

Cuando llamamos `calcularPromedio(1, 2, 3)`, obtenemos un mensaje que dice: 2"

# Resumen 🌈

Entonces, amiguitos, una función en JavaScript es como una receta de cocina. La declaramos con `function`, le damos "ingredientes" llamados parámetros, y a veces nos da algo a cambio llamado "valor de retorno". ¡Es como magia para hacer que nuestro código sea más organizado y divertido!

¡Espero que hayan disfrutado de nuestra clase sobre funciones en JavaScript! Ahora salgan y hagan magia con su código. 🚀✨

# Scope en JavaScript:

El "**scope**" o ámbito en JavaScript se refiere a la región del código en la que una variable es válida y puede ser utilizada. Define las reglas que determinan cómo y dónde las variables pueden ser referenciadas. En JavaScript, hay dos tipos principales de scope: scope global y scope local.

1. **Global Scope:**

- Las variables declaradas fuera de cualquier función o bloque de código tienen un alcance global.

- Estas variables son accesibles desde cualquier parte del código, ya sea dentro de funciones, bloques o incluso en otros archivos si están en el mismo ámbito global.

var globalVariable = 'Soy global';

function verGlobalVariable () {

console.log(globalVariable)

}

verGlobalVariable()

2. **Local Scope:**

- Las variables declaradas dentro de una función o bloque de código tienen un alcance local.

- Estas variables son accesibles solo dentro de la función o bloque en el que fueron declaradas.

function exampleFunction() {

var localVariable = 'Soy local';

console.log(localVariable);

}

- En ES6 (ECMAScript 2015) se introdujo `let` y `const` que tienen un alcance de bloque, lo que significa que su alcance se limita al bloque en el que fueron declarados.

if (true) {

let blockVariable = 'Soy de bloque';

console.log(blockVariable);

}

3. **Lexical Scope:**

- JavaScript utiliza lexical scope, también conocido como "scope estático".

- En lexical scope, el alcance de una variable está determinado por su posición en el código fuente durante la fase de compilación, no durante la ejecución.

- Esto significa que el alcance de una variable se determina por su ubicación dentro del código fuente y la estructura de bloques anidados.

function outerFunction() {

var outerVariable = 'Exterior';

function innerFunction() {

console.log(outerVariable); // Accede a la variable del ámbito exterior

}

innerFunction();

}

En resumen, el scope en JavaScript define la visibilidad y accesibilidad de las variables. Las variables pueden tener alcance global, local o de bloque, y el alcance está determinado por la ubicación y estructura del código en el momento de la compilación.