3.7 Clase en vivo - Profesores

Unidad 3: Funciones y Parámetros

Objetivos de clase

- Ventajas del uso de Funciones
- Parámetros
- Scope - Variables locales y globales
- Funciones anónimas y de tipo flecha

Repaso Unidad 3

Duración: 5 minutos

Consigna: Repasar lo visto en Unidad 3

Ejemplo en vivo: Funciones

Duración: 5 minutos

Consigna: Se le sugiere compartir pantalla y mostrar el siguiente ejemplo:

```
function solicitarNombre() {
let nombreIngresado = prompt("Ingresar nombre")
alert("El nombre ingresado es " + nombreIngresado)
}
solicitarNombre();
```

Tips: Las principales ventajas del uso de funciones son

- Evita instrucciones duplicadas (Principio DRY).
- Soluciona un problema complejo usando tareas sencillas (Principio KISS).

- Focaliza tareas prioritarias para el programa (Principio YAGNI).
- Aporta ordenamiento y entendimiento al código.
- Aporta facilidad y rapidez para hacer modificaciones.

Ejemplo en vivo: Función Sumar, mostrar y calculadora

Ejemplo: Sumar y mostrar

Duración: 5 minutos

Consigna: Se le sugiere compartir pantalla y mostrar el siguiente ejemplo, remarcando la diferencia de parámetro en la declaración y llamado de la función:

```
//Declaración de variable para guardar el resultado de la suma

let resultado = 0;

//Función que suma dos números y asigna a resultado

function sumar(*primerNumero*, *segundoNumero*) {

resultado = *primerNumero* + *segundoNumero*

}

//Función que muestra resultado por consola

function mostrar(*mensaje*) {

console.log(*mensaje*)

}

//Llamamos primero a sumar y luego a mostrar
```

```
sumar(6, 3);

mostrar(resultado);
```

Resultado de una función

Como acabamos de ver, la función puede comportarse como una ***operación*** que genera valores.

En el espacio donde se llama a la función se genera un nuevo valor: este valor es el definido por el **return** de la misma.

let resultado = sumar(5, 8);

console.log(resultado) // ⇒ 13

Ejemplo: Calculadora

Duración: 5 minutos

Consigna: Se le sugiere compartir pantalla y mostrar el siguiente ejemplo:

```
function calculadora(*primerNumero*, *segundoNumero*, *operacion

switch (*operacion*) {

case "+":

return *primerNumero* + *segundoNumero*;

break;

case "-":

return *primerNumero* - *segundoNumero*;

break;

case "*":
```

```
return *primerNumero* * *segundoNumero*;

break;

case "/":

return *primerNumero* / *segundoNumero*;

break;

default:

return 0;

}

}

console.log(calculadora(10, 5, "*"));
```

Actividad: Debugging - Corrigiendo una Función

Duración: 10 minutos

Instrucciones:

- Muestra cómo debuggear y corregir errores en funciones.
- Explica el uso de `return` en funciones para devolver valores.
- Ejemplo de solución:

```
function suma(num1, num2) {
    return num1 + num2;
}
console.log(suma(5, 3));  // Debe mostrar 8
```

Ejemplo en vivo: Scope: Variables globales y locales

Duración: 10 minutos

Consigna: Se presentará un ejemplo en vivo sobre el uso de Scope: variables globales y locales.

Variables Globales

Si una variable se declara fuera de cualquier función o bloque, automáticamente se transforma en variable global.

```
let resultado = 0

function sumar(*primerNumero*, *segundoNumero*) {

resultado = *primerNumero* + *segundoNumero*;

}

sumar(5,6);

//Se puede acceder a la variable resultado porque es global

console.log(resultado);
```

Variables Locales

Cuando definimos una variable dentro de una función o bloque es una variable local, y será accesible sólo dentro de ese espacio. Si queremos utilizarla por fuera, la variable no existirá para JS.

```
function sumar(*primerNumero*, *segundoNumero*) {

let resultado = *primerNumero* + *segundoNumero*;
```

```
}

//No se puede acceder a la variable resultado fuera del bloque

console.error(resultado);
```

Scope

Entender que cada scope local es un espacio cerrado nos permite crear bloques de trabajo bien diferenciados e independientes, sin preocuparnos por repetir nombres de variables, sabiendo que se entienden como diferentes según donde las llamemos.

```
function sumar(*num1*, *num2*) {

let resultado = *num1* + *num2*

 **return resultado

}

function restar(*num1*, *num2*) {

let resultado = *num1* - *num2*

 **return resultado

}
```

Se sugiere enfatizar las ventajas de trabajar con funciones y scope local

Actividad: Explorando el Scope - Global vs. Local

Duración: 15 minutos

Instrucción:

- Explica el concepto de variables globales y locales.
- Discute cómo el scope afecta la accesibilidad de las variables.
- Ejemplo de solución:

```
let color = "rojo";
function cambiarColor() {
    let color = "azul";
}
cambiarColor();
console.log(color);  // Debe mostrar "rojo", porque el `co
lor` dentro de la función es una variable local
```

Ejemplo en vivo

Duración: 10 minutos

Consigna: Imaginemos un sistema de turnos médicos, queremos recorrer un ciclo FOR a partir de una función que recibe un parámetro numérico, el cual será quien defina cuántas veces iterará el ciclo para mostrar turnos disponibles para este mes: ¿Cómo podríamos lograrlo?

Ejemplo en vivo: Funciones Anónimas y Flecha

Duración: 10 minutos

Consigna: En el siguiente ejemplo en vivo se apreciará el uso de las funciones anónimas y flecha.

Funciones Anónimas

Una función anónima es una función que se define sin nombre y se utiliza para ser pasada como parámetro o asignada a una variable. En el caso de asignarla a una variable, pueden llamar usando el identificador de la variable declarada.

```
//Generalmente, las funciones anónimas se asignan a variables de

const suma  = function (*a*, *b*) { return *a* + *b* }

const resta = function (*a*, *b*) { return *a* - *b* }

console.log( suma(15,20) )

console.log( resta(15,5) )
```

Funciones Flecha

Identificamos a las funciones flechas como funciones anónimas de sintaxis simplificada. Están disponibles desde la versión ES6 de JavaScript, no usan la palabra function pero usa ⇒ (flecha) entre los parámetros y el bloque.

```
const suma  = (*a*, *b*) => *a* + *b*;

//Si es una función de una sola línea con retorno podemos evitar

escribir el cuerpo.

const resta = (*a*, *b*) =>  *a* - *b*;

console.log( suma(15,20) )

console.log( resta(20,5) )
```

Ejemplo en vivo: Aplicando Calcular precio

Duración: 5 minutos

Consigna: En el siguiente ejemplo en vivo, se podrá observar el uso de calcular precio.

Se sugiere:

- Remarcar cómo se pasan los llamados de funciones por parámetro
- Sobre ventajas de utilizar funciones flecha referir al material complementario de clase

Ejemplo

const suma = (*a*,*b*) ⇒ *a* + *b*

const resta = (*a*,*b*) ⇒ *a* - *b*

//Si una función flecha puede resolver su objetivo en una sola línea de código y, a su vez, debe retornar ese dato, podemos obviar las llaves **{}** y la palabra reservada **return**

const iva = *x* ⇒ *x* * 0.21

let precioProducto = 500

let descuento = 50

//Calculo: precioProducto + iva - descuento

let nuevoPrecio = resta(suma(precioProducto, iva(precioProducto)), descuento)

console.log(nuevoPrecio)

Ejemplo en vivo

Duración: 10 minutos

Consigna: Si tenemos un negocio y todos los meses compramos productos para vender pero por catálogo nos entregan el precio sin IVA, ¿Cómo podríamos agregar el iva a los productos?

Teniendo la constante IVA = 1.21, imagina una función flecha la cual reciba un valor numérico como parámetro, y que retorne dicho valor con IVA incluido.

El IVA debe tomarse de una constante, para que la función sea dinámica. ¿Cómo te imaginas el resultado?

Actividad: FindTheBug

Duración: 10 minutos

Consigna: Analizaremos dos códigos para hallar el error que está causando un defecto.

Momento de la clase que puede servir de repaso

1. Consultar a los/as estudiantes si están teniendo dificultades para avanzar con alguna parte en el código
2. Preguntar a los/as estudiantes que hayan contestado afirmativamente si quieren compartir pantalla para guiarlo colaborativamente en la resolución del conflicto.
3. La clase (Profesor/a a cargo, tutores/as, coordinador/a y **estudiantes**) leerá el código del/a compañero/a en busca del error que está ocasionando el defecto.
4. Solicitar a los/as estudiantes que propongan soluciones para arreglar el error

Actividad: Completa la función

Duración: 20 minutos

Resolución: Una vez resuelto el ejercicio es importante hacerles recordar a los estudiantes lo siguiente:

// Para evitar errores debes comentar el código donde defines la función **sumar** o cambiar el nombre de la función a **sumar2**, no pueden existir dos funciones, variables o contantes con el mismo nombre.

```
console.log(sumar(5, 5));

};

return resultado;

let resultado = numero1 + numero2;

const sumar = (numero1, numero2) => {
```

```
// función sumar convertida en arrow function => con parámetros
```

// función sumar convertida en arrow function ⇒ con parámetros

```
const sumar = (numero1, numero2) => {

let resultado = numero1 + numero2;

return resultado;

};

console.log(sumar(5, 5));
```

// Para evitar errores debes comentar el código donde defines la función **sumar** o cambiar el nombre de la función a **sumar2**, no pueden existir dos funciones, variables o contantes con el mismo nombre.

```
// Mismo código optimizado

const sumar = (numero1, numero2) => numero1 + numero2;

console.log(sumar(5, 5));

**Respuesta del ejercicio:**

// función sumar

function sumar() {

let numero1 = 5;

let resultado = numero1 + 5;
```

```
return resultado;

}

console.log(sumar());
```

Preclase - 30 minutos previos

Repaso de Conceptos Clave (10 minutos)

- **Funciones en JavaScript:**
  - Repaso rápido sobre la declaración y uso de funciones.
  - Diferencias entre funciones declaradas, expresadas y funciones flecha.
  - Importancia de los parámetros y cómo se utilizan.
- **Scope de Variables:**
  - Explicación breve sobre el ámbito de las variables: global y local.
  - Ejemplos de cómo el scope afecta la accesibilidad de las variables en diferentes bloques de código.
- **Buenas Prácticas en JavaScript:**
  - Resumen de metodologías de programación: DRY (Don't Repeat Yourself), KISS (Keep It Simple, Stupid!), YAGNI (You Aren't Gonna Need It).
  - Consejos para mantener el código limpio y organizado.

Espacio para Preguntas y Respuestas (15 minutos)

- Incentivar a los estudiantes a hacer preguntas sobre los conceptos repasados.
- Resolver dudas de manera clara y concisa, proporcionando ejemplos de código cuando sea necesario.
- Si no hay preguntas, iniciar un diálogo sobre errores comunes y cómo evitarlos al trabajar con funciones y scopes.

Demostraciones Prácticas (Opcional, dependiendo de las preguntas)

- Si surgen dudas que requieran clarificación mediante ejemplos prácticos, realizar demostraciones en vivo.
    - Ejemplos posibles incluyen cómo definir y llamar funciones, y demostrar efectos prácticos del scope en ejecuciones de código.