Funciones y Objetos en JavaScript enfocado al curso de Desarrollo de Aplicaciones Web.

Tema: Funciones y Objetos en JavaScript

Objetivo General:
Al finalizar este tema, los estudiantes serán capaces de crear, utilizar y manipular funciones y objetos en JavaScript, lo que les permitirá estructurar y organizar su código de manera más eficiente para aplicaciones web.

Contenido:

1. Introducción a las Funciones en JavaScript
- Definición de función
- Ventajas del uso de funciones:
  - Reutilización de código
  - Mantenimiento más sencillo
  - Modularidad
- Sintaxis básica:

```
function nombreDeLaFuncion(parametro1, parametro2) {
    // Código a ejecutar
}
```

- Invocación de funciones:
- Llamada a una función:

```
nombreDeLaFuncion(valor1, valor2);
```

- Ejemplo básico:

```
function saludar(nombre) {
    console.log("Hola, " + nombre);
}

saludar("Carlos"); // Salida: Hola, Carlos
```

2. Funciones con valores de retorno
- Uso de `return` para devolver valores de una función:

```
function sumar(a, b) {
    return a + b;
}

let resultado = sumar(5, 3); // resultado será 8
```

- Ejemplo práctico con cálculo de áreas:

```
function areaCuadrado(lado) {
  return lado * lado;
}

let area = areaCuadrado(4); // Salida: 16
```

3. Funciones anónimas y funciones como expresiones

- Definición de funciones anónimas:

```
let sumar = function(a, b) {
  return a + b;
};

console.log(sumar(2, 3)); // Salida: 5
```

- Función autoejecutable (IIFE):

```
(function() {
  console.log("Esta es una función autoejecutable");
})();
```

4. Ámbito de las variables y el concepto de Closures

- Ámbito de función:
  - Variables locales y globales.

```
function ejemplo() {
  let local = "Soy local";
  console.log(local);
}

ejemplo(); // Salida: Soy local
console.log(local); // Error: local no está definida
```

- Closures: Funciones que "recuerdan" el ámbito en el que fueron creadas.

```
function crearContador() {
  let contador = 0;
  return function() {
    contador++;
    return contador;
  };
}

let contador = crearContador();
console.log(contador()); // Salida: 1
```

```
console.log(contador()); // Salida: 2
```

5. Introducción a los Objetos en JavaScript

- Definición de objeto:
  - Colección de propiedades y métodos.
  - Sintaxis básica:

```
let persona = {
    nombre: "Juan",
    edad: 30,
    saludar: function() {
        console.log("Hola, soy " + this.nombre);
    }
};

persona.saludar(); // Salida: Hola, soy Juan
```

6. Propiedades y Métodos de los Objetos

- Propiedades: Características de un objeto.
```
console.log(persona.nombre); // Salida: Juan
```
- Métodos: Funciones asociadas a un objeto.
```
persona.saludar(); // Salida: Hola, soy Juan
```

7. Constructores de Objetos

- Definición de constructores para crear múltiples objetos similares.

```
function Persona(nombre, edad) {
    this.nombre = nombre;
    this.edad = edad;
    this.saludar = function() {
        console.log("Hola, soy " + this.nombre);
    };
}

let persona1 = new Persona("María", 25);
let persona2 = new Persona("Carlos", 35);

persona1.saludar(); // Salida: Hola, soy María
persona2.saludar(); // Salida: Hola, soy Carlos
```

8. Prototipos en JavaScript

- Introducción al concepto de prototype.
- Uso del prototype para agregar métodos a los objetos.

```
Persona.prototype.despedirse = function() {
  console.log(this.nombre + " dice adiós.");
```

9. Clases en JavaScript (ES6)

- Nueva sintaxis para crear objetos usando clases:

```
class Persona {
  constructor(nombre, edad) {
    this.nombre = nombre;
    this.edad = edad;
  }

  saludar() {
    console.log("Hola, soy " + this.nombre);
  }
}

let persona3 = new Persona("Lucía", 28);
persona3.saludar(); // Salida: Hola, soy Lucía
```

10. Herencia en JavaScript

- Herencia usando la sintaxis de clases:

```
class Estudiante extends Persona {
  constructor(nombre, edad, curso) {
    super(nombre, edad); // Llamada al constructor de la clase padre
    this.curso = curso;
  }

  mostrarCurso() {
    console.log(this.nombre + " está en el curso de " + this.curso);
  }
}

let estudiante = new Estudiante("Pedro", 20, "Matemáticas");
estudiante.saludar(); // Salida: Hola, soy Pedro
estudiante.mostrarCurso(); // Salida: Pedro está en el curso de Matemáticas
```

Ejemplo: Aplicación de Gestión de Tareas

Objetivo:
Crear una aplicación que permita agregar, eliminar y listar tareas utilizando funciones y objetos.

HTML - Estructura de la interfaz:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Gestión de Tareas</title>
</head>
<body>
  <h1>Gestión de Tareas</h1>

  <div>
    <label for="nuevaTarea">Agregar una nueva tarea:</label>
    <input type="text" id="nuevaTarea" placeholder="Escribe tu tarea">
    <button onclick="agregarTarea()">Agregar</button>
  </div>

  <h2>Lista de Tareas</h2>
  <ul id="listaTareas">
    <!-- Las tareas se mostrarán aquí -->
  </ul>

  <script src="script.js"></script>
</body>
</html>
```

JavaScript - Funciones y objetos para gestionar tareas:

```
// Clase Tarea para crear objetos tarea
class Tarea {
  constructor(nombre) {
    this.nombre = nombre;
    this.completada = false;
  }

  // Método para completar la tarea
  completar() {
```

```
    this.completada = true;
  }

  // Método para mostrar la tarea con su estado
  mostrarInfo() {
    return `${this.nombre} - ${this.completada ? "Completada" : "Pendiente"}`;
  }
}

// Arreglo para almacenar las tareas
let listaDeTareas = [];

// Función para agregar una tarea
function agregarTarea() {
  let inputTarea = document.getElementById('nuevaTarea');
  let nombreTarea = inputTarea.value.trim();

  if (nombreTarea !== "") {
    // Crear un nuevo objeto Tarea
    let nuevaTarea = new Tarea(nombreTarea);
    // Agregar la tarea al arreglo
    listaDeTareas.push(nuevaTarea);
    // Limpiar el input
    inputTarea.value = "";
    // Mostrar la lista actualizada de tareas
    mostrarTareas();
  } else {
    alert("Por favor, escribe una tarea.");
  }
}

// Función para mostrar todas las tareas en el DOM
function mostrarTareas() {
  let listaTareasElemento = document.getElementById('listaTareas');
  listaTareasElemento.innerHTML = "";

  // Iterar sobre la lista de tareas y crear elementos <li>
  listaDeTareas.forEach((tarea, indice) => {
    let tareaElemento = document.createElement('li');
    tareaElemento.textContent = tarea.mostrarInfo();

    // Crear botón para eliminar la tarea
    let botonEliminar = document.createElement('button');
    botonEliminar.textContent = "Eliminar";
```

```
      botonEliminar.onclick = function() {
        eliminarTarea(indice);
      };

      // Crear botón para marcar como completada
      let botonCompletar = document.createElement('button');
      botonCompletar.textContent = "Completar";
      botonCompletar.onclick = function() {
        completarTarea(indice);
      };

      // Agregar botones al <li>
      tareaElemento.appendChild(botonCompletar);
      tareaElemento.appendChild(botonEliminar);

      // Agregar el <li> al <ul>
      listaTareasElemento.appendChild(tareaElemento);
    });
}

// Función para eliminar una tarea
function eliminarTarea(indice) {
    listaDeTareas.splice(indice, 1); // Eliminar la tarea del arreglo
    mostrarTareas(); // Mostrar la lista actualizada
}

// Función para marcar una tarea como completada
function completarTarea(indice) {
    listaDeTareas[indice].completar(); // Marcar la tarea como completada
    mostrarTareas(); // Mostrar la lista actualizada
}
```

Explicación del Código:

1. HTML:
  - El formulario contiene un campo de texto (`input`) para ingresar una nueva tarea y un botón para agregarla.
  - Las tareas se mostrarán dentro de un elemento `<ul>` (lista no ordenada) con el id `listaTareas`.

2. JavaScript:
  - Clase `Tarea`: Se utiliza para definir las tareas. Cada tarea tiene un `nombre` y un estado (`completada` o no completada).

- El método `completar()` cambia el estado de la tarea a completada.
- El método `mostrarInfo()` devuelve una cadena con el nombre de la tarea y su estado actual.

- Funciones:
  - `agregarTarea()`: Toma el valor del campo de texto, crea una nueva tarea y la agrega a la lista de tareas.
  - `mostrarTareas()`: Muestra todas las tareas en el DOM. Se crean elementos `<li>` para cada tarea, con botones para eliminar o completar.
  - `eliminarTarea(indice)`: Elimina una tarea del arreglo `listaDeTareas` según su índice.
  - `completarTarea(indice)`: Marca una tarea como completada.

Ejemplo en Acción:

1. Agregar una tarea: El usuario escribe el nombre de la tarea en el campo de texto y presiona el botón "Agregar". Esto ejecuta la función `agregarTarea()`, que crea un nuevo objeto de la clase `Tarea` y lo agrega a la lista de tareas.

2. Mostrar las tareas: Después de agregar una tarea, la función `mostrarTareas()` recorre la lista de tareas y las muestra en el navegador, creando un elemento de lista (`<li>`) para cada tarea, junto con los botones de "Completar" y "Eliminar".

3. Eliminar o completar una tarea: Los botones de cada tarea permiten al usuario eliminar la tarea o marcarla como completada. La función `eliminarTarea()` elimina la tarea del arreglo, y la función `completarTarea()` cambia su estado a completada.

Actividad Adicional:

- Modificar la aplicación para que las tareas completadas se muestren en una sección separada de las tareas pendientes.
- Agregar la funcionalidad de editar una tarea existente.
- Guardar las tareas en el `localStorage` del navegador para que las tareas persistan después de recargar la página.