Carrera: Tecnología en Desarrollo de Software

Nivel: I I I Semestre

Nombre del Estudiante: Segura Pesantes Edson Gabriel

Asignatura: DISEÑO WEB

Docente: ING. ERNESTO ROBLES

2024 2025

Investigación sobre estructuras y funciones en JavaScript

1. Conversión implícita y explícita de tipos en JavaScript

JavaScript es un lenguaje débilmente tipado, lo que significa que puede convertir valores entre tipos automáticamente (conversión implícita) o mediante funciones explícitas.

- **Conversión implícita:** JavaScript cambia el tipo automáticamente según el contexto, por ejemplo, al sumar un número con una cadena, el número se convierte en cadena.

```
let resultado1 = '5' + 3;   // El número 3 se convierte a string y se concatena
console.log(resultado1);    // Salida: '53'

let resultado2 = '5' * 3;   // La cadena '5' se convierte a número y se multiplica
console.log(resultado2);    // Salida: 15
```

- **Conversión explícita:** El programador usa funciones para transformar tipos, como se usa cuando el programador indica el cambio de tipo con funciones especiales.

```
let numero = Number('123');       // Convierte string a número
console.log(numero);              // Salida: 123

let texto = String(456);          // Convierte número a string
console.log(texto);               // Salida: '456'

let booleano = Boolean(0);        // Convierte número a booleano (0 es false)
console.log(booleano);            // Salida: false
```

2. Ejemplos de estructuras condicionales y repetitivas

Las estructuras condicionales permiten tomar decisiones según condiciones, y las repetitivas permiten repetir bloques de código.

- Condicional if - else:

```
let edad = 18;
if (edad >= 18) {
    console.log("Eres mayor de edad");
} else {
    console.log("Eres menor de edad");
}
```

- Condicional switch:

```
let dia = 2;
switch(dia) {
    case 1:
        console.log("Lunes");
        break;
    case 2:
        console.log("Martes");
        break;
    default:
        console.log("Otro día");
}
```

Repetitivas

- **For**

```
for(let i = 0; i < 3; i++) {
    console.log("Número:", i);
}
```

- **While**

```
let contador = 0;
while(contador < 3) {
    console.log("Contador:", contador);
    contador++;
}
```

3. Arreglos y objetos

Declaración y acceso a elementos

```
let frutas = ['manzana', 'banana', 'pera'];
console.log(frutas[1]); // banana

let persona = {nombre: 'Ana', edad: 25};
console.log(persona.nombre);  // Ana
console.log(persona['edad']); // 25
```

Métodos comunes de arreglos

```
// map: crea un nuevo arreglo con el resultado de aplicar una función a cada elemento
let numeros = [1, 2, 3];
let cuadrados = numeros.map(x => x * x);
console.log(cuadrados); // [1, 4, 9]

// filter: crea un nuevo arreglo con elementos que cumplen la condición
let mayoresQueUno = numeros.filter(x => x > 1);
console.log(mayoresQueUno); // [2, 3]

// concat: une dos arreglos en uno nuevo (no modifica los originales)
let otrosNumeros = [4, 5];
let combinado = numeros.concat(otrosNumeros);
console.log(combinado); // [1, 2, 3, 4, 5]
```

4.- Coloca un ejemplo para cada tipo de función:

-Funciones declaradas

```
function saludar(nombre) {
    return `Hola, ${nombre}`;
}
console.log(saludar('Juan'));
```

-Funciones como expresiones anónimas

```
const multiplicar = function(a, b) {
    return a * b;
};
console.log(multiplicar(2, 3));
```

-Función Flecha

```
const dividir = (a, b) => a / b;
console.log(dividir(10, 2));
```

-Función callbacks

```
function procesarArreglo(arr, callback) {
    for(let i = 0; i < arr.length; i++) {
        callback(arr[i]);
    }
}

procesarArreglo([1, 2, 3], elemento => console.log(elemento * 2));
```

5. Eventos más comunes usados para llamar funciones

Los eventos en JavaScript permiten ejecutar funciones como respuesta a acciones del usuario. A continuación, se enumeran y explican brevemente los más utilizados:

- **`onclick`:** Se dispara al hacer clic en un elemento. Muy común en botones.

```
<button onclick="alert('¡Hola!')">Clic aquí</button>
```

•**`oninput`**: Se activa mientras se escribe o cambia el valor de un campo de entrada.

```
<input type="text" oninput="console.log(this.value)">
```

• **`onchange`**: Ocurre cuando se modifica un valor y luego se pierde el enfoque del campo.

```
<select onchange="console.log(this.value)">
  <option value="A">A</option>
  <option value="B">B</option>
</select>
```

• **`onsubmit`**: Se activa al enviar un formulario.

```
<form onsubmit="alert('Formulario enviado'); return false;">
  <button type="submit">Enviar</button>
</form>
```

• **`onmouseover`**: Se ejecuta cuando el puntero pasa sobre un elemento.

```
<div onmouseover="console.log('Cursor encima')">Pasa el mouse</div>
```

6. Buenas prácticas en la escritura del código

- **Nombres significativos**: Las variables y funciones deben tener nombres claros y descriptivos.

```
let temperaturaCelsius = 25;
function convertirACelsius(fahrenheit) {
  return (fahrenheit - 32) * 5 / 9;
}
```

Comentarios útiles: Explican el propósito del código, especialmente en bloques complejos.

```
// Calcula el área de un rectángulo a partir de base y altura
function calcularArea(base, altura) {
  return base * altura;
}
```

Funciones reutilizables: Separar el código en funciones pequeñas y específicas evita duplicación y facilita pruebas y cambios.

```
function saludar(nombre) {
  return `Hola, ${nombre}`;
}

console.log(saludar("Ana"));
console.log(saludar("Luis"));
```

Bibliografía

• Mozilla Developer Network (MDN Web Docs). JavaScript Guide. Recuperado de: https://developer.mozilla.org/es/docs/Web/JavaScript/Guide

• W3Schools. JavaScript Tutorial. Recuperado de: https://www.w3schools.com/js/

• Haverbeke, Marijn. Eloquent JavaScript (3ª ed.). 2018. ISBN: 9781593279509

• JavaScript.info. The Modern JavaScript Tutorial. Recuperado de: https://javascript.info

• FreeCodeCamp. JavaScript Algorithms and Data Structures Certification. Recuperado de: https://www.freecodecamp.org/learn