Capítulo 2: Fundamentos de JavaScript Creativo

En este capítulo aprenderás los ingredientes básicos de JavaScript que necesitas para empezar a programar cualquier tipo de proyecto web. Aprenderás cómo funcionan las variables, los tipos de datos, las estructuras de control y las funciones en JavaScript. Además, aprenderás a utilizar la consola del navegador para depurar tus programas y a escribir comentarios en tu código para hacerlo más legible.

Variables, tipos de datos y operadores

Expresiones y declaraciones (Expressions and Statements)

La programación en JavaScript se basa en la creación y manipulación de valores mediante expresiones y declaraciones. Una **expresión** es una combinación de valores, variables y operadores que se evalúa para producir un resultado. Por ejemplo, la expresión `2 + 3` evalúa a `5`.

Una **declaración** es una instrucción que realiza una acción. Por ejemplo, la declaración `alert('Hola, mundo!')` muestra una ventana emergente con el mensaje "Hola, mundo!".

```
// Expresión
2 + 3;

// Declaración
alert('Hola, mundo!');
```

Variables (Variables)

Una **variable** es la referencia a un valor que puede cambiar a lo largo del tiempo. En JavaScript, las variables se declaran con las siguientes palabras clave: `var`, `let` y `const`. Cabe destacar que las palabras clave, son aquellas palapras que forman parte del lenguaje de programación y que tienen un significado especial.

- `var`: Se utilizaba para declarar variables en versiones antiguas de JavaScript. No se recomienda su uso en la actualidad.
- `let`: Se utiliza para declarar variables que pueden cambiar su valor.

- `const`: Se utiliza para declarar variables que no pueden cambiar su valor.

```
// Declaración de variables

let nombre = 'Juan'; // Variable que almacena un valor de tipo
string

let edad = 30; // Variable que almacena un valor de tipo number

let esEstudiante = true; // Variable que almacena un valor de tipo
boolean
```

Constantes (Constants)

Las **constantes** son variables cuyo valor no puede cambiar una vez que se ha asignado. En JavaScript, se utilizan la palabra clave `const` para declarar constantes.

```
// Declaración de constantes
const PI = 3.141592653589793; // Constante que almacena el valor de
PI

const URL = 'https://www.ejemplo.com'; // Constante que almacena
una URL
```

Hoy en día, se recomienda utilizar `let` y `const` en lugar de `var` para declarar variables en JavaScript. La diferencia entre `let` y `const` radica en que `let` permite cambiar el valor de la variable, mientras que `const` no lo permite.

Tipos de datos (Data Types)

En JavaScript, existen varios tipos de datos que puedes utilizar para almacenar valores. Algunos de los tipos de datos más comunes son:

- `string`: Cadena de texto.
- `number`: Número.
- `boolean`: Valor booleano (`true` o `false`).
- `null`: Valor nulo.
- `undefined`: Valor indefinido.
- `object`: Objeto.
- `array`: Arreglo.

```
// Tipos de datos

let nombre = 'Juan'; // Tipo de dato: string

let edad = 30; // Tipo de dato: number

let esEstudiante = true; // Tipo de dato: boolean

let persona = { nombre: 'Juan', edad: 30 }; // Tipo de dato: object

let numeros = [1, 2, 3, 4, 5]; // Tipo de dato: array
```

Nota: JavaScript es un lenguaje de programación de tipado dinámico, lo que significa que no es necesario especificar el tipo de dato de una variable al declararla. El tipo de dato de una variable se determina automáticamente en tiempo de ejecución.

Strings (Cadenas de texto)

Las **cadenas de texto** (`string`) son secuencias de caracteres que se utilizan para representar texto en JavaScript. Puedes crear cadenas de texto utilizando comillas simples (`'`) o dobles (`"`) y por último, las comillas invertidas (`) para cadenas de texto de varias líneas.

- Comillas simples: `'Hola, mundo!'`
- Comillas dobles: `"Hola, mundo!"`
- Comillas invertidas: `` `Hola, mundo!` `` (Template literals) (`Alt + 96`)

```
// Cadenas de texto

let nombre = 'Juan'; // Cadena de texto con comillas simples

let saludo = "Hola, mundo!"; // Cadena de texto con comillas dobles

let mensaje = `Bienvenido a mi página web`; // Cadena de texto con
comillas invertidas
```

Salto de línea en cadenas de texto

Para incluir saltos de línea en una cadena de texto en JavaScript, puedes utilizar el carácter de escape `\n`.

```
// Salto de línea en cadenas de texto

let mensaje = 'Hola,\nMundo!';

console.log(mensaje); // Salida: Hola,
                      //         Mundo!
```

Concatenación de cadenas (String concatenation)

La **concatenación de cadenas** es la operación de unir dos o más cadenas de texto en una sola cadena. En JavaScript, puedes concatenar cadenas de texto utilizando el operador `+`.

```
// Concatenación de cadenas

let nombre = 'Juan';

let saludo = 'Hola, ' + nombre + '!

console.log(saludo); // Salida: Hola, Juan!
```

Literales de plantilla (Template literals)

Los **literales de plantilla** (`template literals`) son una forma de crear cadenas de texto de forma más legible y concisa en JavaScript. Los literales de plantilla se crean utilizando comillas invertidas (`) y permiten incrustar expresiones de JavaScript dentro de la cadena de texto utilizando `${}`.

```
// Literales de plantilla

let nombre = 'Juan';

let edad = 30;

let mensaje = `Hola, me llamo ${nombre} y tengo ${edad} años.`;

console.log(mensaje); // Salida: Hola, me llamo Juan y tengo 30
años.
```

Nota: Los literales de plantilla son una forma más moderna y legible de crear cadenas de texto en JavaScript. Se recomienda utilizarlos en lugar de las concatenaciones de cadenas con el operador `+`.

Números (Numbers)

Los **números** (`number`) se utilizan para representar valores numéricos en JavaScript. Los números pueden ser enteros (por ejemplo, `1`, `2`, `3`) o decimales (por ejemplo, `1.5`, `3.14`, `10.5`).

```
// Números

let edad = 30; // Número entero

let precio = 19.99; // Número decimal

let cantidad = 3; // Número entero
```

Operaciones aritméticas (Arithmetic operations)

En JavaScript, puedes realizar operaciones aritméticas con números utilizando operadores como `+` (suma), `-` (resta), `*` (multiplicación), `/` (división) y `%` (módulo).

```
// Operaciones aritméticas

let suma = 2 + 3; // Suma: 5

let resta = 5 - 2; // Resta: 3

let multiplicacion = 2 * 3; // Multiplicación: 6

let division = 6 / 2; // División: 3

let modulo = 5 % 2; // Módulo: 1
```

Conmutación de operadores (Operator precedence)

En JavaScript, las operaciones aritméticas siguen un orden de precedencia estándar. Puedes utilizar paréntesis `()` para cambiar el orden de evaluación de las operaciones.

```
// Conmutación de operadores

let resultado = 2 + 3 * 4; // 14

let resultado2 = (2 + 3) * 4; // 20
```

Incremento y decremento (Increment and decrement)

En JavaScript, puedes incrementar (`++`) o decrementar (`--`) el valor de una variable numérica en una unidad.

```
// Incremento y decremento

let contador = 0;

contador++; // Incremento en 1: 1

contador--; // Decremento en 1: 0
```

Booleanos (Booleans)

Los valores **booleanos** (`boolean`) representan la lógica de verdad en JavaScript. Los valores booleanos pueden ser `true` (verdadero) o `false` (falso).

```
// Booleanos

let esEstudiante = true; // Verdadero

let esMayorDeEdad = false; // Falso
```

Nota: Los valores booleanos son útiles para controlar el flujo de un programa utilizando estructuras de control como `if`, `else`, `while`, `for`, entre otras que veremos más adelante.

💡 **Nota de color**: Los valores booleanos reciben su nombre del matemático Boole, quien desarrolló el álgebra booleana, un sistema matemático que se utiliza para representar la lógica de verdad en la programación.

Nulos y no definidos (Null and Undefined)

Los valores **nulos** (`null`) y **no definidos** (`undefined`) son dos valores especiales en JavaScript que representan la ausencia de un valor.

- `null`: Representa la ausencia intencional de un valor.
- `undefined`: Representa la ausencia no intencional de un valor.

```
// Nulos y no definidos

let valorNulo = null; // Valor nulo

let valorNoDefinido; // Valor no definido
```

Nota: Es importante tener en cuenta la diferencia entre `null` y `undefined` en JavaScript. `null` se utiliza para representar la ausencia intencional de un valor, mientras que `undefined` se utiliza para representar la ausencia no intencional de un valor.

Arrays (Arreglos) y Objetos (Objects)

Estos son dos tipos de datos compuestos en JavaScript que te permiten almacenar múltiples valores en una sola variable. Por lo tanto, ambos son a su vez estructuras de datos. Es por eso que abordaremos estos dos tipos de datos en detalle más adelante.

Operadores (Operators)

Los **operadores** son símbolos especiales que se utilizan para realizar operaciones en JavaScript. Los operadores se utilizan para realizar operaciones aritméticas, de comparación, lógicas, de asignación, entre otras.

Operadores aritméticos (Arithmetic operators)

Los **operadores aritméticos** se utilizan para realizar operaciones matemáticas en JavaScript. Algunos de los operadores aritméticos más comunes son:

- `+`: Suma
- `-`: Resta
- `*`: Multiplicación
- `/`: División
- `%`: Módulo (resto de la división)

```
// Operadores aritméticos

let suma = 2 + 3; // Suma: 5

let resta = 5 - 2; // Resta: 3

let multiplicacion = 2 * 3; // Multiplicación: 6

let division = 6 / 2; // División: 3

let modulo = 5 % 2; // Módulo: 1
```

Operadores de comparación (Comparison operators)

Los **operadores de comparación** se utilizan para comparar dos valores en JavaScript y devolver un valor booleano (`true` o `false`). Algunos de los operadores de comparación más comunes son:

- `==`: Igual a
- `===`: Estrictamente igual a
- `!=`: Diferente de
- `!==`: Estrictamente diferente de
- `>`: Mayor que
- `<`: Menor que
- `>=`: Mayor
- `<=`: Menor o igual que

```
// Operadores de comparación

let a = 5;

let b = 3;

console.log(a == b); // Salida: false
```

```
console.log(a === b); // Salida: false

console.log(a != b); // Salida: true

console.log(a !== b); // Salida: true

console.log(a > b); // Salida: true

console.log(a < b); // Salida: false

console.log(a >= b); // Salida: true

console.log(a <= b); // Salida: false
```

Operadores lógicos (Logical operators)

Los **operadores lógicos** se utilizan para combinar o invertir valores booleanos en JavaScript. Algunos de los operadores lógicos más comunes son:

- `&&`: Y (AND)
- `||`: O (OR)
- `!`: No (NOT)

```
// Operadores lógicos

console.log( 5<3 && 3>2 ); // Salida: false (false && true = false)
console.log(5=='5'&& 3<2); // Salida: false (true && false = false)
console.log(5<3 && 5==='5'); // Salida: false (false && false =
false)
console.log(5>3 && 3>2); // Salida: true (true && true = true)

console.log(5<3 || 3>2); // Salida: true (false || true = true)
console.log(5=='5' || 3<2); // Salida: true (true || false = true)
console.log(5>3 || 3>2); // Salida: true (true || true = true)
console.log(5<3 || 5==='5'); // Salida: false (false || false =
false)

// operador de negación
```

```
console.log(!true); // Salida: false
console.log(!false); // Salida: true
```

Operadores de asignación (Assignment operators)

Los **operadores de asignación** se utilizan para asignar valores a variables en JavaScript. Algunos de los operadores de asignación más comunes son:

- `=`: Asignación
- `+=`: Suma y asignación
- `-=`: Resta y asignación
- `*=`: Multiplicación y asignación
- `/=`: División y asignación
- `%=`: Módulo y asignación

```
// Operadores de asignación

let x = 5;

x += 3; // x = x + 3

console.log(x); // Salida: 8

x -= 2; // x = x - 2
console.log(x); // Salida: 6


x *= 2; // x = x * 2
console.log(x); // Salida: 12

x /= 3; // x = x / 3

console.log(x); // Salida: 4

x %= 2; // x = x % 2
```

```
console.log(x); // Salida: 0
```

Nota: Los operadores de asignación son útiles para realizar operaciones aritméticas y asignar el resultado a una variable en una sola instrucción.

Comentarios (Comments)

Los **comentarios** son fragmentos de texto que se utilizan para documentar y explicar el código en JavaScript. Los comentarios no se ejecutan y no afectan el funcionamiento del programa, pero son útiles para hacer que el código sea más legible y comprensible.

En JavaScript, existen dos tipos de comentarios:

- Comentarios de una sola línea: Se crean utilizando `//` y se extienden hasta el final de la línea.
- Comentarios de varias líneas: Se crean utilizando `/*` para abrir el comentario y `*/` para cerrarlo.

```
// Comentario de una sola línea

/*

Comentario de
varias líneas

*/
```

Los comentarios son una buena práctica de programación que te ayudará a documentar tu código y a explicar su funcionamiento a otras personas que puedan leerlo en el futuro.

Depuración (Debugging)

La **depuración** es el proceso de identificar y corregir errores en un programa. En JavaScript, puedes utilizar la consola del navegador para depurar tus programas y ver mensajes de error, advertencia e información.

Para imprimir mensajes en la consola del navegador, puedes utilizar el método `console.log()`.

```
// Depuración

let nombre = 'Juan';

let edad = 30;

console.log(nombre); // Imprime el valor de la variable nombre en la consola

console.log(edad); // Imprime el valor de la variable edad en la consola
```

Además de `console.log()`, la consola del navegador también te permite utilizar otros métodos como `console.error()`, `console.warn()`, `console.info()`, entre otros, para imprimir mensajes de error, advertencia e información en la consola.

```
// Depuración

let nombre = 'Juan';

let edad = 30;

console.error('Error: No se pudo cargar el archivo'); // Imprime un mensaje de error en la consola

console.warn('Advertencia: La contraseña es débil'); // Imprime un mensaje de advertencia en la consola

console.info('Información: El tiempo de carga es de 2 segundos');
// Imprime un mensaje de información en la consola
```

La consola del navegador es una herramienta poderosa que te permitirá depurar tus programas y ver mensajes de error, advertencia e información en tiempo real.

Actividades

Vamos a poner en práctica lo que has aprendido en este capítulo con las siguientes actividades desde un enfoque creativo:

1. Crea una variable `nombre` y asígnale tu nombre.
2. Crea una variable `edad` y asígnale tu edad.
3. Crea una variable `esEstudiante` y asígnale `true` si eres estudiante o `false` si no lo eres.
4. Crea una variable `mensaje` que contenga un saludo personalizado utilizando literales de plantilla.
5. Imprime el mensaje en la consola del navegador.

Para realizar esta actividad, podés utilizar la carpeta `practica` que se encuentra en la carpeta de este capítulo. Descarga la carpeta `practica` a tu computadora, ábrela en tu editor de código (Visual Studio Code) y sigue los pasos indicados en el archivo `index.html`. O también podés ingresar a este enlace para realizar la actividad en línea.

Recordá hacer un Fork del proyecto para que puedas realizar cambios y guardar tu progreso en **StackBlitz**.

Estructuras de control y bucles

Las **estructuras de control** y los **bucles** son elementos fundamentales en la programación que te permiten controlar el flujo de un programa y repetir tareas de forma eficiente. En esta sección, aprenderás cómo utilizar estructuras de control como `if`, `else`, `switch` y bucles como `for`, `while` y `do...while` en JavaScript.

Estructuras de control (Control structures)

Las **estructuras de control** te permiten controlar el flujo de un programa y tomar decisiones basadas en condiciones específicas. Algunas de las estructuras de control más comunes en JavaScript son:

- `if`: Se utiliza para ejecutar un bloque de código si se cumple una condición.
- `else`: Se utiliza para ejecutar un bloque de código si no se cumple la condición del `if`.
- `else if`: Se utiliza para evaluar múltiples condiciones en un solo bloque de código.
- `switch`: Se utiliza para evaluar múltiples casos y ejecutar un bloque de código basado en el caso que se cumpla.
- `? :` (operador ternario): Se utiliza para evaluar una condición y devolver un valor basado en la condición

if...else

La estructura `if...else` se utiliza para ejecutar un bloque de código si se cumple una condición y otro bloque de código si no se cumple la condición.

```
// Estructura if...else

let edad = 18;

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

else if

La estructura `else if` se utiliza para evaluar múltiples condiciones en un solo bloque de código.

```
// Estructura else if

let hora = 14;

if (hora < 12) {
    console.log('Buenos días');
} else if (hora < 18) {
    console.log('Buenas tardes');
} else {
    console.log('Buenas noches');
}
```

switch

La estructura `switch` se utiliza para evaluar múltiples casos y ejecutar un bloque de código basado en el caso que se cumpla.

```
// Estructura switch

let dia = 3;

switch (dia) {
    case 1:
        console.log('Lunes');
        break;
    case 2:
        console.log('Martes');
        break;
    case 3:
        console.log('Miércoles');
        break;
    case 4:
        console.log('Jueves');
```

```
        break;
    case 5:
        console.log('Viernes');
        break;
    case 6:
        console.log('Sábado');
        break;
    case 7:
        console.log('Domingo');
        break;
    default:
        console.log('Día inválido');
}
```

Operador ternario

El operador ternario `? :` se utiliza para evaluar una condición y devolver un valor basado en la condición.

Sintaxis

```
condición ? valorSiVerdadero : valorSiFalso;
```

Ejemplo

```
// Operador ternario

let edad = 18;

let mensaje = edad >= 18 ? 'Eres mayor de edad' : 'Eres menor de edad';

console.log(mensaje);
```

Nota: El operador ternario es una forma abreviada de escribir una estructura `if...else` en una sola línea y es parte del Sugar Syntax de JavaScript o en otros términos, azúcar sintáctica, una forma más sencilla de escribir código.

Bucles (Loops)

En programación muchas veces es necesario repetir una instrucción o un bloque de instrucciones varias veces. Por ejemplo, imaginemos que queremos imprimir los números del 1 al 10 en la consola. Podríamos hacerlo de la siguiente manera:

```
console.log(1);
console.log(2);
console.log(3);
console.log(4);
console.log(5);
console.log(6);
console.log(7);
console.log(8);
console.log(9);
console.log(10);
```

Bien, la implementación de este código no es una tarea difícil de hacer, sin embargo, si queremos imprimir los números del 1 al 100, la tarea se vuelve más tediosa y repetitiva. Es aquí donde entran en juego los bucles, que nos permiten repetir una instrucción o un bloque de instrucciones varias veces de forma eficiente.

En JavaScript, existen varios tipos de bucles que te permiten repetir tareas de forma eficiente:

- `for`: Se utiliza para repetir una instrucción o un bloque de instrucciones un número específico de veces.
- `while`: Se utiliza para repetir una instrucción o un bloque de instrucciones mientras se cumpla una condición.
- `do...while`: Se utiliza para repetir una instrucción o un bloque de instrucciones al menos una vez y luego mientras se cumpla una condición.
- `for...in`: Se utiliza para iterar sobre las propiedades de un objeto.

- `for...of`: Se utiliza para iterar sobre los elementos de un objeto iterable (como un array).

for

El bucle `for` se utiliza para repetir una instrucción o un bloque de instrucciones un número específico de veces.

Sintaxis

```
for (inicialización; condición; actualización) {
    // Bloque de código a repetir
}
```

Ejemplo

```
// Bucle for

for (let i = 1; i <= 10; i++) {
    console.log(i);
}
```

while

El bucle `while` se utiliza para repetir una instrucción o un bloque de instrucciones mientras se cumpla una condición. Es importante tener en cuenta que la condición debe ser verdadera para que el bucle se ejecute.

Sintaxis

```
while (condición) {
    // Bloque de código a repetir
}
```

Ejemplo

```
// Bucle while

let i = 1;

while (i <= 10) {
    console.log(i);
    i++;
}
```

do...while

El bucle `do...while` se utiliza para repetir una instrucción o un bloque de instrucciones al menos una vez y luego mientras se cumpla una condición. A diferencia del bucle `while`, el bucle `do...while` ejecuta el bloque de código al menos una vez antes de evaluar la condición.

Sintaxis

```
do {
    // Bloque de código a repetir
} while (condición);
```

Ejemplo

```
// Bucle do...while

let i = 1;

do {
    console.log(i);
    i++;
} while (i <= 10);
```

Actividades

Vamos a poner en práctica lo que has aprendido en este capítulo con las siguientes actividades desde un enfoque creativo:

1. Crea un bucle `for` que imprima los números del 1 al 10 en la consola.
2. Crea un blucle `for` que imprima solo los números pares del 1 al 10 en la consola.
3. Crea un bucle `for` que imprima "bizz" si el número es divisible por 3, "buzz" si el número es divisible por 5 y "bizzbuzz" si el número es divisible por 3 y 5. De lo contrario, imprime el número.