5. Números (Numbers)

Los números son un tipo de dato fundamental en cualquier lenguaje de programación, y JavaScript no es la excepción. En este capítulo, exploraremos cómo trabajar con números en JavaScript, incluyendo su creación, operaciones matemáticas, propiedades y métodos útiles. Además, aprenderás a manejar números decimales, enteros y valores especiales como `NaN` e `Infinity`.

Creación de Variables Numéricas

En JavaScript, los números pueden ser enteros o decimales, y no es necesario declarar explícitamente su tipo. Puedes crear variables numéricas simplemente asignando un valor.

Ejemplo:

```
let entero = 42;
let decimal = 3.14;
```

Notación Científica

JavaScript también permite representar números muy grandes o muy pequeños utilizando notación científica.

Ejemplo:

```
let numeroGrande = 1.23e5; // 123000
let numeroPequeño = 1.23e-5; // 0.0000123
```

Operadores Aritméticos y su Precedencia

Los **operadores aritméticos** se utilizan para realizar operaciones matemáticas básicas como suma, resta, multiplicación y división.

Operadores Aritméticos Comunes:

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

Ejemplo:

```
let a = 10;
let b = 3;

console.log(a + b); // 13
```

```
console.log(a - b); // 7
console.log(a * b); // 30
console.log(a / b); // 3.333...
console.log(a % b); // 1 (resto de 10 ÷ 3)
console.log(a ** b); // 1000 (10 elevado a 3)
```

Precedencia de Operadores

La **precedencia** determina el orden en que se evalúan las operaciones en una expresión. Los operadores con mayor precedencia se evalúan primero. Puedes usar paréntesis para forzar un orden específico.

Orden de Precedencia:

1. Paréntesis `()`
2. Exponenciación `*`
3. Multiplicación , División `/`, Módulo `%`
4. Suma `+`, Resta

Ejemplo:

```
let resultado = 10 + 5 * 2; // Multiplicación tiene mayor precedencia
console.log(resultado); // 20

resultado = (10 + 5) * 2; // Paréntesis cambian el orden
console.log(resultado); // 30
```

Operadores Relacionales

Los **operadores relacionales** comparan dos valores y devuelven un valor booleano (`true` o `false`). Son útiles en estructuras condicionales y bucles.

Operadores Relacionales Comunes:

- `>`: Mayor que
- `<`: Menor que
- `>=`: Mayor o igual que
- `<=`: Menor o igual que
- `==`: Igualdad (sin considerar el tipo)
- `===`: Igualdad estricta (considera el tipo)
- `!=`: Desigualdad (sin considerar el tipo)
- `!==`: Desigualdad estricta (considera el tipo)

Ejemplo:

```
let a = 5;
let b = "5";

console.log(a > 3); // true
console.log(a == b); // true (compara solo el valor)
console.log(a === b); // false (compara valor y tipo)
console.log(a != b); // false
console.log(a !== b); // true
```

Operadores de Incremento y Decremento

Los **operadores de incremento (`++`) y decremento (`--`)** aumentan o disminuyen el valor de una variable en 1.

Formas:

- **Prefijo**: El operador se aplica antes de que se evalúe la variable.
- **Postfijo**: El operador se aplica después de que se evalúe la variable.

Ejemplo:

```
let x = 5;

console.log(++x); // 6 (incremento prefijo)
console.log(x++); // 6 (incremento postfijo, pero se incrementa después)
console.log(x);   // 7

let y = 10;
console.log(--y); // 9 (decremento prefijo)
console.log(y--); // 9 (decremento postfijo, pero se decrementa después)
console.log(y);   // 8
```

Operador Unario

Un **operador unario** es un operador que actúa sobre un solo operando. En JavaScript, algunos ejemplos incluyen:

- `+`: Convierte un valor a número.
- : Convierte un valor a número y lo niega.
- `typeof`: Devuelve el tipo de dato de un valor.
- `delete`: Elimina una propiedad de un objeto.

Ejemplo:

```
let texto = "123";
console.log(+texto); // 123 (conversión a número)
```

```
let numero = 42;
console.log(-numero); // -42

console.log(typeof "hola"); // string
console.log(typeof 42);     // number

let persona = { nombre: "Ana", edad: 25 };
delete persona.edad;
console.log(persona); // { nombre: "Ana" }
```

Valores Especiales

JavaScript incluye algunos valores especiales relacionados con los números:

1. `NaN` **(Not-a-Number)**:
   Representa un valor numérico inválido, generalmente resultado de operaciones incorrectas.

   ```
   let resultado = "hola" * 2; // NaN
   console.log(isNaN(resultado)); // true
   ```

2. `Infinity` y `Infinity`:
   Representan números infinitamente grandes (positivos o negativos).

   ```
   console.log(1 / 0); // Infinity
   console.log(-1 / 0); // -Infinity
   ```

Propiedades y Métodos Útiles

JavaScript proporciona varias herramientas para trabajar con números de manera más precisa y eficiente.

Propiedades del Objeto `Number`

1. `Number.MAX_VALUE`:
   El mayor número representable en JavaScript.

   ```
   console.log(Number.MAX_VALUE); // 1.7976931348623157e+308
   ```

2. `Number.MIN_VALUE`:
   El menor número positivo representable.

   ```
   console.log(Number.MIN_VALUE); // 5e-324
   ```

3. `Number.NaN`:
   Representa el valor
   `NaN`.

```
console.log(Number.NaN); // NaN
```

4. `Number.POSITIVE_INFINITY` y `Number.NEGATIVE_INFINITY`:
   Representan infinito positivo y negativo.

```
console.log(Number.POSITIVE_INFINITY); // Infinity
```

Métodos del Objeto `Number`

1. `toFixed(decimales)`:
   Redondea un número a una cantidad específica de decimales y lo convierte en un string.

```
let precio = 123.456;
console.log(precio.toFixed(2)); // "123.46"
```

2. `toPrecision(totalDigitos)`:
   Formatea un número con una cantidad total de dígitos significativos.

```
let numero = 123.456;
console.log(numero.toPrecision(4)); // "123.5"
```

3. `parseInt(string)`:
   Convierte un string en un número entero.

```
console.log(parseInt("42")); // 42
console.log(parseInt("42.99")); // 42
```

4. `parseFloat(string)`:
   Convierte un string en un número decimal.

```
console.log(parseFloat("42.99")); // 42.99
```

5. `isNaN(valor)`:
   Verifica si un valor es `NaN`.

```
console.log(isNaN("hola")); // true
console.log(isNaN(42)); // false
```

6. `isFinite(valor)`:
   Verifica si un valor es un número finito.

```
console.log(isFinite(42)); // true
console.log(isFinite(Infinity)); // false
```

Math: El Objeto para Operaciones Matemáticas Avanzadas

El objeto global `Math` proporciona una colección de propiedades y métodos para realizar cálculos matemáticos avanzados.

Propiedades Comunes

1. `Math.PI`:
   El valor de π (pi).

   ```
   console.log(Math.PI); // 3.141592653589793
   ```

2. `Math.E`:
   El número de Euler (base de los logaritmos naturales).

   ```
   console.log(Math.E); // 2.718281828459045
   ```

Métodos Comunes

1. `Math.round(numero)`:
   Redondea un número al entero más cercano.

   ```
   console.log(Math.round(4.5)); // 5
   ```

2. `Math.floor(numero)`:
   Redondea hacia abajo al entero más cercano.

   ```
   console.log(Math.floor(4.9)); // 4
   ```

3. `Math.ceil(numero)`:
   Redondea hacia arriba al entero más cercano.

   ```
   console.log(Math.ceil(4.1)); // 5
   ```

4. `Math.random()`:
   Genera un número aleatorio entre 0 (inclusive) y 1 (exclusivo).

   ```
   console.log(Math.random()); // Ejemplo: 0.23456789
   ```

5. `Math.max(...numeros)`:
   Devuelve el número más grande de una lista.

   ```
   console.log(Math.max(1, 5, 3, 9)); // 9
   ```

6. `Math.min(...numeros)`:
   Devuelve el número más pequeño de una lista.

   ```
   console.log(Math.min(1, 5, 3, 9)); // 1
   ```

7. `Math.pow(base, exponente)`:
   Eleva un número a una potencia.

   ```
   console.log(Math.pow(2, 3)); // 8
   ```

8. `Math.sqrt(numero)`:
   Calcula la raíz cuadrada de un número.

   ```
   console.log(Math.sqrt(16)); // 4
   ```

Conversión de Strings a Números

A menudo, necesitarás convertir strings en números, especialmente cuando trabajas con entradas de usuario. JavaScript proporciona varias formas de hacerlo:

1. `Number(string)`:
   Convierte un string en un número.

   ```
   console.log(Number("42")); // 42
   console.log(Number("42.99")); // 42.99
   ```

2. **Operador Unario** `+`:
   También puedes usar el operador
   `+` para convertir un string en un número.

   ```
   console.log(+"42"); // 42
   ```

3. `parseInt` **y** `parseFloat`:
   Ya mencionados anteriormente, estos métodos son útiles para casos específicos.
