Number

Number as a Data Type

JavaScript mein **Number** ek **primitive data type** hai, jo integers aur floating-point numbers (decimal numbers) dono ko represent karta hai.

Examples of Number

```
let integerNumber = 42; // Simple integer
let floatingPoint = 3.14; // Decimal number
let negativeNumber = -10; // Negative integer
```

Special Numeric Values

1. **Infinity**: Jab koi number divide hota hai 0 se ya bohot bara hota hai.

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

2. **Infinity**: Jab koi negative number divide hota hai 0 se.

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

3. **NaN (Not-a-Number)**: Jab mathematical operation invalid ho.

   ```
   console.log("Hello" / 2); // NaN
   ```

Number Methods

JavaScript mein `Number` object ke bohot saare methods available hain jo numbers ko manipulate karte hain. Neeche inka zikr hai:

1. Number.parseInt()

String ko integer number mein convert karta hai.

```
let num = Number.parseInt("42");
console.log(num); // 42
```

2. Number.parseFloat()

String ko floating-point number mein convert karta hai.

```
let num = Number.parseFloat("3.14");
console.log(num); // 3.14
```

3. toFixed()

Decimal places ko fix karta hai aur string return karta hai.

```
let num = 3.14159;
console.log(num.toFixed(2)); // "3.14"
```

4. toString()

Number ko string mein convert karta hai.

```
let num = 255;
console.log(num.toString()); // "255"
```

5. Number.isFinite()

Check karta hai ke value finite hai ya nahi.

```
console.log(Number.isFinite(10)); // true
console.log(Number.isFinite(Infinity)); // false
```

6. Number.isInteger()

Check karta hai ke value ek integer hai ya nahi.

```
console.log(Number.isInteger(10)); // true
console.log(Number.isInteger(10.5)); // false
```

7. Number.isNaN()

Check karta hai ke value NaN hai ya nahi.

```
console.log(Number.isNaN(NaN)); // true
console.log(Number.isNaN(10)); // false
```

8. Number.MAX_VALUE

JavaScript mein possible sabse bara number.

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

The largest number that can be represented in JavaScript. Equal to approximately 1.79E+308.

9. Number.MIN_VALUE

JavaScript mein possible sabse chhota positive number.

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

10. Number.EPSILON

2 numbers ke darmiyan minimum difference ka represent karta hai.

```
(Number.EPSILON); // 2.220446049250313e-16
```

11. Number.MAX_SAFE_INTEGER

Sabse bara integer jo safely represent kiya ja sakta hai.

```
console.log(Number.MAX_SAFE_INTEGER); // 9007199254740991
```

12. Number.MIN_SAFE_INTEGER

Sabse chhota integer jo safely represent kiya ja sakta hai.

```
```

```
console.log(Number.MIN_SAFE_INTEGER);
```

Type Conversion

JavaScript mein ek data type ko dusre mein convert karne ke liye **explicit** aur **implicit conversions** hoti hain.

Explicit Conversion

1. `Number(value)`:

```
console.log(Number("123")); // 123
```

2. `parseInt(value)` aur `parseFloat(value)` bhi use hote hain.

Implicit Conversion

Jab JavaScript automatic conversion karti hai.

```
let result = "5" * 2; // 10
```

Common Operations

1. **Arithmetic Operations**:

```
let sum = 5 + 3; // 8
let difference = 5 - 3; // 2
let product = 5 * 3; // 15
let quotient = 5 / 2; // 2.5
```

```
let remainder = 5 % 2; // 1
```

2. **Exponents**:

```
console.log(2 ** 3); // 8
```

3. **Increment/Decrement**:

```
let x = 5;
x++;
console.log(x); // 6
```

Interesting Facts

1. JavaScript mein numbers 64-bit **IEEE 754** format mein store hote hain.
2. Floating-point operations **precision loss** ka shikar ho sakte hain:

```
console.log(0.1 + 0.2); // 0.30000000000000004
```

3. Isko fix karne ke liye `Number.EPSILON` ka istemal hota hai.

JavaScript mein numbers **64-bit IEEE 754 floating-point format** ke mutabiq store hote hain. Yeh ek widely used standard hai jo decimal aur binary numbers ko store aur calculate karne ka tareeqa batata hai. Lekin is format ki wajah se kuch **limitations aur precision issues** aate hain.

Precision Loss Problem

IEEE 754 floating-point format ke limitation ki wajah se **precision loss** hota hai, jo majorly floating-point operations mein dikhta hai.

Example of Precision Loss:

```
console.log(0.1 + 0.2); // 0.30000000000000004
```

Yahaan expected result `0.3` hai, lekin rounding errors ki wajah se JavaScript mein yeh value thodi si inaccurate ho jati hai.

Use `Number.EPSILON`:

`Number.EPSILON` ko comparison ke liye use kiya jata hai, jo smallest possible difference define karta hai.

```
let a = 0.1 + 0.2;
let b = 0.3;

console.log(Math.abs(a - b) < Number.EPSILON); // true
```

2. Convert to Integers:

Floating-point operations se bachne ke liye numbers ko integers mein convert karein.

```
let a = (0.1 * 10) + (0.2 * 10); // 3
console.log(a / 10); // 0.3
```