Numbers, Dates, Intl & Timers

> 🎓 **CS Perspective:** JavaScript uses **IEEE 754 double-precision floating-point** for all numbers, which explains precision issues like `0.1 + 0.2 !== 0.3`. This is a fundamental limitation of binary floating-point representation—certain decimal fractions cannot be exactly represented. BigInt provides **arbitrary-precision integers** when you need exact large number arithmetic. Timers (`setTimeout`, `setInterval`) are part of the **event loop** mechanism—they don't execute immediately when the timer expires but are queued in the callback/task queue, demonstrating JavaScript's **non-preemptive concurrency model**. The Intl API showcases **internationalization (i18n)** concerns in software engineering.

📌 Numbers in JavaScript

All numbers are floating-point (no integers):

```
console.log(23 === 23.0); // true

// Floating point precision issues
console.log(0.1 + 0.2);           // 0.30000000000000004
console.log(0.1 + 0.2 === 0.3);   // false ⚠
```

📌 Converting Strings to Numbers

```
// Number function
Number('23');    // 23

// Unary plus (shortcut)
+'23';           // 23

// Parsing
Number.parseInt('30px', 10);     // 30
Number.parseInt('e23', 10);      // NaN
Number.parseFloat('2.5rem');     // 2.5
```

📌 Checking Numbers

```
// isNaN - Check if NOT a number
Number.isNaN(20);        // false
Number.isNaN('20');      // false
Number.isNaN(+'20X');    // true
Number.isNaN(23 / 0);    // false (Infinity)
```

```
// isFinite - Best for checking if value is a number
Number.isFinite(20);     // true ✅
Number.isFinite('20');   // false
Number.isFinite(23 / 0); // false

// isInteger
Number.isInteger(23);    // true
Number.isInteger(23.0);  // true
Number.isInteger(23.5);  // false
```

📌 Math and Rounding

Math Methods

```
Math.sqrt(25);       // 5
25 ** (1/2);         // 5 (same as sqrt)
8 ** (1/3);          // 2 (cube root)

Math.max(5, 18, 23); // 23
Math.min(5, 18, 23); // 5

Math.PI;             // 3.141592653589793

Math.random();       // 0 to < 1
Math.trunc(Math.random() * 6) + 1; // 1-6
```

Random Integer in Range

```
const randomInt = (min, max) =>
  Math.floor(Math.random() * (max - min + 1)) + min;

randomInt(10, 20); // 10-20 inclusive
```

Rounding Integers

```
Math.round(23.3);  // 23 (nearest)
Math.round(23.9);  // 24

Math.ceil(23.3);   // 24 (up)
Math.ceil(23.9);   // 24

Math.floor(23.3);  // 23 (down)
Math.floor(23.9);  // 23
```

```
Math.trunc(23.3);  // 23 (remove decimal)
Math.trunc(-23.3); // -23
Math.floor(-23.3); // -24 ⚠ Different for negatives!
```

Rounding Decimals

```
(2.7).toFixed(0);    // '3' (string!)
(2.7).toFixed(3);    // '2.700'
(2.345).toFixed(2);  // '2.35'
+(2.345).toFixed(2); // 2.35 (number)
```

📌 Remainder Operator (%)

```
5 % 2;   // 1 (5 = 2 * 2 + 1)
8 % 3;   // 2 (8 = 3 * 2 + 2)

// Check even/odd
const isEven = n => n % 2 === 0;
isEven(8);  // true
isEven(23); // false

// Every nth element
if (i % 2 === 0) // every 2nd
if (i % 3 === 0) // every 3rd
```

📌 Numeric Separators (ES2021)

```
const diameter = 287_460_000_000; // 287460000000
const price = 345_99;             // 34599
const fee = 15_00;                // 1500

// ⚠ Don't work with strings
Number('230_000'); // NaN
```

📌 BigInt (ES2020)

For integers larger than `2^53 - 1`:

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

// Create BigInt
```

```
const huge = 20289830237283728378237n;
const alsoHuge = BigInt(48384302);

// Operations
10000n + 10000n;  // 20000n
huge * 10000n;    // Works

// ⚠ Can't mix with regular numbers
huge * 23;        // Error!
huge * BigInt(23); // Works

// Comparisons
20n > 15;   // true
20n === 20; // false (different types)
20n == '20'; // true (type coercion)

// Division truncates
11n / 3n;   // 3n (not 3.666...)
```

📌 Creating Dates

```
// Current date/time
const now = new Date();

// From string
new Date('Aug 02 2020 18:05:41');
new Date('December 24, 2015');

// From components (month is 0-indexed!)
new Date(2037, 10, 19, 15, 23, 5);
// Nov 19, 2037 15:23:05

// From timestamp (ms since Jan 1, 1970)
new Date(0);                      // Jan 1, 1970
new Date(3 * 24 * 60 * 60 * 1000); // 3 days later
```

📌 Date Methods

```
const future = new Date(2037, 10, 19, 15, 23);

future.getFullYear();  // 2037
future.getMonth();     // 10 (November, 0-indexed)
future.getDate();      // 19 (day of month)
future.getDay();       // 4 (day of week, 0 = Sunday)
future.getHours();     // 15
future.getMinutes();   // 23
future.getSeconds();   // 0
```

```
future.toISOString();  // '2037-11-19T14:23:00.000Z'
future.getTime();      // Timestamp in ms

// Set methods
future.setFullYear(2040);

// Current timestamp
Date.now(); // Milliseconds since Jan 1, 1970
```

📌 Date Calculations

```
// Dates can be subtracted (returns milliseconds)
const future = new Date(2037, 10, 19);
console.log(+future); // Timestamp

const calcDaysPassed = (date1, date2) =>
  Math.abs(date2 - date1) / (1000 * 60 * 60 * 24);

calcDaysPassed(new Date(2037, 3, 4), new Date(2037, 3, 14));
// 10 days
```

📌 Intl - Internationalization API

Dates

```
const now = new Date();

// Basic
new Intl.DateTimeFormat('en-US').format(now); // 12/17/2025

// With options
const options = {
  hour: 'numeric',
  minute: 'numeric',
  day: 'numeric',
  month: 'long', // 'numeric', '2-digit', 'long', 'short'
  year: 'numeric',
  weekday: 'long'
};

new Intl.DateTimeFormat('en-US', options).format(now);
// "Wednesday, December 17, 2025 at 3:45 PM"

// Use browser locale
new Intl.DateTimeFormat(navigator.language, options).format(now);
```

Numbers

```
const num = 3884764.23;

const options = {
  style: 'currency', // 'decimal', 'percent', 'currency', 'unit'
  currency: 'EUR',
  // unit: 'celsius' // for style: 'unit'
};

new Intl.NumberFormat('en-US', options).format(num);
// €3,884,764.23

new Intl.NumberFormat('de-DE', options).format(num);
// 3.884.764,23 €
```

📌 setTimeout

Executes once after delay:

```
setTimeout(() => console.log('Hello'), 3000);
// Logs "Hello" after 3 seconds

// With arguments
setTimeout((ing1, ing2) => {
  console.log(`Pizza with ${ing1} and ${ing2}`);
}, 3000, 'olives', 'spinach');

// Cancel timer
const timer = setTimeout(() => console.log('Hello'), 3000);
clearTimeout(timer);
```

📌 setInterval

Executes repeatedly:

```
// Every second
setInterval(() => {
  const now = new Date();
  console.log(now);
}, 1000);

// Stop interval
const timer = setInterval(callback, 1000);
clearInterval(timer);
```

📌 Countdown Timer Pattern

```
const startTimer = function() {
  const tick = function() {
    const min = String(Math.trunc(time / 60)).padStart(2, 0);
    const sec = String(time % 60).padStart(2, 0);

    labelTimer.textContent = `${min}:${sec}`;

    if (time === 0) {
      clearInterval(timer);
      // Logout user
    }

    time--;
  };

  let time = 120; // 2 minutes
  tick(); // Call immediately
  const timer = setInterval(tick, 1000);

  return timer;
};
```

🧪 Quick Reference Cheatsheet

```
// Conversions
+'23';                    // String to number
Number.parseInt('30px');  // Parse integer
Number.parseFloat('2.5rem'); // Parse float

// Checking
Number.isFinite(x);       // Is it a number?
Number.isInteger(x);      // Is it an integer?

// Rounding
Math.round(x);  // Nearest
Math.ceil(x);   // Up
Math.floor(x);  // Down
Math.trunc(x);  // Remove decimal
x.toFixed(2);   // Returns string!

// Random
Math.random();                    // 0–1
Math.trunc(Math.random() * n) + 1; // 1–n
```

```
// Dates
new Date();
date.getFullYear();
date.getMonth();   // 0-indexed!
date.getDate();
date.getTime();    // Timestamp
Date.now();        // Current timestamp

// Intl
new Intl.DateTimeFormat(locale, options).format(date);
new Intl.NumberFormat(locale, options).format(num);

// Timers
setTimeout(fn, ms);    // Once
setInterval(fn, ms);   // Repeated
clearTimeout(timer);
clearInterval(timer);
```

✅ Exam Tips

1. **All JS numbers are floats** - `23 === 23.0`
2. **0.1 + 0.2 !== 0.3** - Floating point precision
3. `Number.isFinite()` is best for checking if value is a number
4. **Month is 0-indexed** in `new Date(year, month, day)`
5. `Math.floor(-23.3)` = -24, `Math.trunc(-23.3)` = -23
6. `toFixed()` returns a **string**
7. **BigInt** can't mix with regular numbers without conversion
8. `setTimeout` is async - code continues immediately
9. Clear timers before setting new ones to avoid duplicates
10. Use `padStart(2, 0)` for two-digit formatting
11. `navigator.language` gets browser's locale
12. Subtract dates to get milliseconds difference