Date parsing in JavaScript

Date parsing in JavaScript is the process of transforming a string into a `Date` object. This operation is fundamental in many applications, for example when receiving data from an API or an HTML form.

Creating a Date Object

The `Date` constructor can be used in several ways. One of the most common is passing it a string:

```
const date = new Date("2025-07-23T10:30:00Z");
console.log(date.toISOString());
```

The ISO 8601 format (`YYYY-MM-DDTHH:mm:ssZ`) is the safest and most consistent for automatic parsing.

Parsing with Date.parse()

You can also use `Date.parse()`, which returns the timestamp (in milliseconds) corresponding to the date:

```
const timestamp = Date.parse("2025-07-23T10:30:00Z");
console.log(new Date(timestamp));
```

Here too, the use of the ISO format is recommended.

Support for Localized Formats

Native JavaScript does not handle localized date formats well (such as `23/07/2025` or `07-23-2025`). These may produce inconsistent results or

`Invalid Date`:

```
const date = new Date("23/07/2025"); // Potentially
invalid
console.log(date.toString());
```

For these cases, it is recommended to use a specialized library.

Advanced Parsing with Libraries

Libraries such as **date-fns**, **Luxon**, and **Day.js** offer robust parsing and support for multiple formats:

```
// With Day.js
dayjs("23/07/2025", "DD/MM/YYYY").toDate();

// With Luxon
luxon.DateTime.fromFormat("23/07/2025",
"dd/MM/yyyy").toJSDate();
```

Best Practices

- Always use the ISO 8601 format when possible.
- Avoid implicit parsing of localized strings.
- Always check the validity of the `Date` object after parsing.
- For complex parsing, use reliable libraries.

Conclusion

Date parsing in JavaScript can be simple if the correct conventions are followed. However, for international applications or complex requirements, using external libraries is often the best choice.