Background Story

It was a muggy Monday morning in Makati City, and Odessa found herself nursing a cup of kapeng barako at a nearby café ☕. Her dev team at the fintech startup was growing—suddenly they had five interns scattered across Luzon, Visayas, and Mindanao, each in different time zones due to remote work! Kuya Ronnie challenged Odessa: “Ods, automate our intern schedules. We need clock-in reminders, shift calendars, and time-difference checks. You in?”

Odessa’s eyes sparkled. She opened her code editor and typed `new Date()`. She felt like Doctor Strange bending time itself. First, she printed the current timestamp:

```
console.log(new Date());
// e.g. 2023-07-12T02:15:30.000Z
```

But next, she needed to show Philippine time, not UTC. She discovered `toLocaleString("en-PH")` and watched her screen:

```
console.log(new Date().toLocaleString("en-PH"));
// e.g. "7/12/2023, 10:15:30 AM"
```

🔥 Game changer. Odessa then computed tomorrow’s date to send “Good morning!” emails before interns log in:

```
const today = new Date();
const tomorrow = new Date(today);
tomorrow.setDate(today.getDate() + 1);
```

Her laptop beeped. A Slack message from Maria, an intern in Davao: “Ma’am, when’s our next weekly sync in my time?” Odessa calculated the time difference with `getTimezoneOffset()` and converted it to hours. Suddenly scheduling across Luzon-Visayas-Mindanao felt easy.

Later, as the sky turned pink over the Ayala skyline, Odessa formatted a PDF calendar for their HR: Monday to Friday shifts at 9 AM–5 PM, with weekends off. She printed dates as `"DD MMM YYYY"` strings:

```
const day = String(d.getDate()).padStart(2, "0");
const month = d.toLocaleString("en-PH", { month: "short" });
const year = d.getFullYear();
console.log(`${day} ${month} ${year}`);
// e.g. "13 Jul 2023"
```

Odessa leaned back. She had bent time with code—and saved her team hours of manual scheduling. From Manila to Zamboanga, everyone now got precise reminders. She imagined herself leading a product team that could automate entire HR workflows for Pinoy companies. All it took was mastering JavaScript's Date object.

Tomorrow, she'd tackle Promises and Async/Await—but for now, she was "clocked in" as the scheduling hero of her startup. 🚀

Theory & Lecture Content

JavaScript's `Date` object lets you work with dates and times. In this lesson, we'll cover:

1. Creating `Date` instances
2. Retrieving components (year, month, day, hours, minutes, seconds)
3. Modifying dates (adding days, months)
4. Calculating time differences
5. Formatting dates into human-readable strings

1. Creating Date Instances

- **Current date/time**:

  ```
  const now = new Date();
  ```

- **Specific date** (year, monthIndex [0-11], day, hour, ...):

  ```
  const specific = new Date(2023, 6, 15, 9, 0, 0);
  // July is monthIndex 6
  ```

- **From timestamp** (milliseconds since Jan 1 1970 UTC):

  ```
  const fromTs = new Date(1689350400000);
  ```

- **From date string**:

  ```
  const fromStr = new Date("2023-07-15T09:00:00");
  ```

Reference: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/Date

---

2. Retrieving Components

Methods to get parts of the date:

```
const d = new Date();
d.getFullYear(); // e.g. 2023
d.getMonth(); // 0-11 (0=Jan)
d.getDate(); // day of month (1-31)
d.getHours(); // 0-23
d.getMinutes(); // 0-59
d.getSeconds(); // 0-59
d.getTime(); // timestamp in ms
d.getTimezoneOffset(); // difference from UTC in minutes
```

---

3. Modifying Dates

Use setters or date math:

```
const d = new Date();

// Add 1 day
d.setDate(d.getDate() + 1);

// Add 2 hours
d.setHours(d.getHours() + 2);

// Move to next month
d.setMonth(d.getMonth() + 1);
```

---

4. Calculating Time Differences

Compute difference in milliseconds, then convert:

```
const start = new Date("2023-07-10T09:00:00");
const end = new Date("2023-07-11T18:30:00");

const diffMs = end.getTime() - start.getTime();
const diffHours = diffMs / (1000 * 60 * 60); // e.g. 33.5 hours
```

---

5. Formatting Dates

A. Manual formatting

```
function formatPH(d) {
  const day = String(d.getDate()).padStart(2, "0");
  const month = d.toLocaleString("en-PH", { month: "short" });
  const year = d.getFullYear();
  const hr = String(d.getHours()).padStart(2, "0");
  const min = String(d.getMinutes()).padStart(2, "0");
  return `${day} ${month} ${year}, ${hr}:${min}`;
}

console.log(formatPH(new Date())); // "12 Jul 2023, 10:15"
```

B. Using `toLocaleString`

```
const opts = {
  year: "numeric",
  month: "short",
  day: "2-digit",
  hour: "2-digit",
  minute: "2-digit",
  hour12: true,
  timeZone: "Asia/Manila",
};
console.log(new Date().toLocaleString("en-PH", opts));
// "12 Jul 2023, 10:15 AM"
```

Reference:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/toLocaleString

---

Exercises

Exercise 1: Tomorrow's Date Formatter

Problem Statement
Implement `getTomorrow()` in `exercise1.js` to return tomorrow's date as a string in `YYYY-MM-DD` format.

TODOs

- Create a `Date` for today.
- Add one day.
- Format to `"YYYY-MM-DD"`.

Starter Code (exercise1.js)

```
function getTomorrow() {
  // TODO: implement
}

module.exports = { getTomorrow };
```

Full Solution (exercise1.js)

```
function getTomorrow() {
  const today = new Date();
  const tmr = new Date(today);
  tmr.setDate(today.getDate() + 1);

  const yyyy = tmr.getFullYear();
  const mm = String(tmr.getMonth() + 1).padStart(2, "0");
  const dd = String(tmr.getDate()).padStart(2, "0");

  return `${yyyy}-${mm}-${dd}`;
}

module.exports = { getTomorrow };
```

Exercise 2: Hours Difference Calculator

Problem Statement
Write `hoursBetween(start, end)` in `exercise2.js` that takes two ISO date strings and returns the difference in hours (decimal allowed).

TODOs

- Parse both strings to `Date`.
- Compute difference in milliseconds.
- Convert to hours, return a number.

Starter Code (exercise2.js)

```
function hoursBetween(start, end) {
  // TODO: implement
}

module.exports = { hoursBetween };
```

Full Solution (exercise2.js)

```
function hoursBetween(start, end) {
  const s = new Date(start);
  const e = new Date(end);
  const diffMs = e.getTime() - s.getTime();
  return diffMs / (1000 * 60 * 60);
}

module.exports = { hoursBetween };
```

Exercise 3: Weekly Schedule Generator

Problem Statement

Implement `generateMondays(startDate, weeks)` in `exercise3.js`. Starting from the given `startDate` string (`YYYY-MM-DD`), return an array of `Date` objects for each Monday for the next `weeks` weeks (including the first Monday on or after `startDate`).

TODOs

- Parse `startDate`.
- Find first Monday (day index 1).
- Loop `weeks` times, push each Monday and add 7 days.

Starter Code (exercise3.js)

```
function generateMondays(startDate, weeks) {
  // TODO: implement
}

module.exports = { generateMondays };
```

Full Solution (exercise3.js)

```
function generateMondays(startDate, weeks) {
  const result = [];
  const d = new Date(startDate);
  // day: 0=Sun,1=Mon,...6=Sat
  const offset = (8 - d.getDay()) % 7; // days to next Monday
  d.setDate(d.getDate() + offset);

  for (let i = 0; i < weeks; i++) {
    result.push(new Date(d));
    d.setDate(d.getDate() + 7);
  }
  return result;
}

module.exports = { generateMondays };
```

Exercise 4: Custom Date Formatter

Problem Statement

In `exercise4.js`, write `formatPHDate(dateObj)` that takes a `Date` object and returns a string `"DD MMM YYYY, hh:mm AM/PM"` in Manila time.

TODOs

- Use `toLocaleString` with options.
- Ensure 12-hour format and Manila timezone.

Starter Code (exercise4.js)

```
function formatPHDate(dateObj) {
  // TODO: implement
}

module.exports = { formatPHDate };
```

Full Solution (exercise4.js)

```
function formatPHDate(dateObj) {
  const opts = {
    day: "2-digit",
    month: "short",
    year: "numeric",
    hour: "2-digit",
    minute: "2-digit",
    hour12: true,
    timeZone: "Asia/Manila",
  };
  return dateObj.toLocaleString("en-PH", opts);
}

module.exports = { formatPHDate };
```

Test Cases

exercise1.test.js

```
const { getTomorrow } = require("./exercise1");

test("returns tomorrow in YYYY-MM-DD", () => {
  const today = new Date("2023-07-12T00:00:00");
  // mock today's date
  jest.spyOn(global, "Date").mockImplementation(() => today);

  expect(getTomorrow()).toBe("2023-07-13");

  global.Date.mockRestore();
});
```

exercise2.test.js

```
const { hoursBetween } = require("./exercise2");

test("calculates hours difference", () => {
  const h = hoursBetween("2023-07-10T09:00:00", "2023-07-10T18:30:00");
  expect(h).toBeCloseTo(9.5);
});
```

exercise3.test.js

```
const { generateMondays } = require("./exercise3");

test("generates correct Mondays", () => {
  const mondays = generateMondays("2023-07-12", 3);
  const isoList = mondays.map((d) => d.toISOString().slice(0, 10));
  expect(isoList).toEqual(["2023-07-17", "2023-07-24", "2023-07-31"]);
});
```

exercise4.test.js

```
const { formatPHDate } = require("./exercise4");

test("formats date in Manila time", () => {
  const date = new Date("2023-07-12T02:15:00Z"); // UTC
  const formatted = formatPHDate(date);
  // Manila is UTC+8 → 10:15 AM
  expect(formatted).toMatch(/12 Jul 2023, 10:15\s?AM/);
});
```

Closing Story

Night had fallen, and the city lights of Makati blinked like a million code editors running tests. Odessa sat at her desk, smiling at the success of her automated schedules. Interns from Baguio to General Santos were already receiving their shift reminders in the morning. She had bent time—both future dates and time differences—into her will.

Kuya Ronnie congratulated her on Slack: “Ods, you just saved HR dozens of hours every week! Next, we dive into asynchronous APIs—Promises & Async/Await. Ready?”

Odessa stretched, filled with excitement. Handling time was magical, but working with real data from servers… that felt like summoning lightning ⚡. She closed her eyes for a moment, picturing herself as a startup CTO, leading a team that moves at the speed of light. Tomorrow, the asynchronous journey begins.

She took a sip of her iced coffee and typed one last line in her journal:
“Time is code—and code is power.”

Her next adventure awaited. 🚀