Background Story

It was a humid Tuesday afternoon in Quezon City, and Odessa was hunched over her laptop in the UP Diliman campus coffee shop ☕. She had just finished debugging a basic to-do list app using ES5 syntax—and it felt like she was speaking in old Tagalog slang. Her tech lead, Kuya Ronnie, dropped by her table with a grin.

“Ods, ready to level up? Let’s learn ES6. Modern syntax, modern mindset!” he said, sliding his own laptop closer.

Odessa’s eyes widened. ES6? She had heard about arrow functions, destructuring, and template strings—but had never tried them herself. Kuya Ronnie showed her a classic function:

```
function sum(a, b) {
  return a + b;
}
```

Then he rewrote it in ES6:

```
const sum = (a, b) => a + b;
```

“Parang English to Taglish, di ba? Shortcut pa,” he laughed. Odessa tried it out. Sinubukan, na-compile, gumana agad. She felt that spark—she was unlocking a secret dialect of JavaScript.

The next day, during their online internship at a Manila-based startup, Odessa’s team was building a weather widget. They had a response object:

```
const weather = {
  city: "Manila",
  temp: 30,
  condition: "Sunny",
};
```

Her teammate was doing this:

```
const city = weather.city;
const temp = weather.temp;
const condition = weather.condition;
```

Kuya Ronnie tapped his keyboard and showed her destructuring:

```
const { city, temp, condition } = weather;
```

Odessa smiled so wide, she almost spilled her tsokolate.

Finally, for showing the forecast message, they were doing:

```
const message =
  "Good morning! Weather in " +
  city +
  " is " +
  temp +
  "°C and " +
  condition +
  ".";
```

He taught her template literals:

```
const message = `Good morning! Weather in ${city} is ${temp}°C and ${condition}.`;
```

It felt so natural—like texting a friend on Viber. Odessa realized these ES6 features made code cleaner, prevented mistakes, and saved time—just like that extra shot of expresso 🏃☕.

As dusk settled over Quezon City, Odessa sat back and imagined herself a year later: leading her own dev team, mentoring juniors, and speaking JavaScript so fluently that new interns would call her “Tita Odessa.” With ES6 in her toolkit, she felt ready to conquer any coding challenge—from barangay apps to nationwide FinTech platforms.

Her journey from ES5 to ES6 was just the beginning. With her heart pounding like a jeepney horn in EDSA traffic, she was ready for the next module: Promises and Async/Await.

Theory & Lecture Content

JavaScript ES6 (ECMAScript 2015) introduced powerful syntax improvements. Today we focus on three key features:

1. Arrow Functions
2. Destructuring (Arrays & Objects)
3. Template Literals

1. Arrow Functions

Traditional function:

```
function add(a, b) {
  return a + b;
}
```

ES6 arrow function:

```
const add = (a, b) => a + b;
```

• If only one parameter, you can omit parentheses:

```
const square = (x) => x * x;
```

• If zero or multiple statements inside, use braces and `return`:

```
const multiply = (a, b) => {
  const result = a * b;
  return result;
};
```

`this` binding
Arrow functions do not have their own `this`. They inherit `this` from the enclosing context → useful in callbacks.

Reference:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

---

2. Destructuring

Object Destructuring

```
const user = { name: "Odessa", age: 20, course: "IT" };
const { name, age } = user;
console.log(name); // "Odessa"
console.log(age); // 20
```

You can also assign new variable names:

```
const { name: fullName, course } = user;
```

Array Destructuring

```
const scores = [95, 88, 76];
const [math, english, science] = scores;
console.log(english); // 88
```

Reference:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment

---

3. Template Literals

Traditional string concatenation:

```
const city = "Cebu";
const temp = 29;
const msg = "Weather in " + city + " is " + temp + "°C.";
```

ES6 Template Literal:

```
const msg = `Weather in ${city} is ${temp}°C.`;
```

You can embed expressions and multi-line strings:

```
const a = 5,
  b = 10;
console.log(`Fifteen is ${a + b} and
not ${2 * a + b}.`);
```

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

---

Exercises

Exercise 1: Refactor to Arrow Functions

Problem Statement
Refactor the traditional functions in `exercise1.js` into arrow functions.

TODOs

- Convert `add` to an arrow function.
- Convert `multiply` to an arrow function with a block body.
- Convert `greet` to a concise arrow function.

Starter Code (exercise1.js)

```
// TODO: Refactor to arrow functions

function add(a, b) {
  return a + b;
}

function multiply(a, b) {
  const result = a * b;
  return result;
}

function greet(name) {
  return "Hello, " + name + "!";
}

module.exports = { add, multiply, greet };
```

Full Solution (exercise1.js)

```
// ES6 Arrow Function Refactoring

const add = (a, b) => a + b;

const multiply = (a, b) => {
  const result = a * b;
  return result;
};

const greet = (name) => `Hello, ${name}!`;

module.exports = { add, multiply, greet };
```

Exercise 2: Use Object Destructuring

Problem Statement

Complete the `getFullName` function in `exercise2.js` by using object destructuring to extract `firstName` and `lastName` from the `person` object.

TODOs

- Inside `getFullName`, use object destructuring.
- Return the full name in the format `"First Last"`.

Starter Code (exercise2.js)

```
```

```
function getFullName(person) {
  // TODO: Use destructuring to extract firstName and lastName
  const first = person.firstName;
  const last = person.lastName;
  return first + " " + last;
}

module.exports = { getFullName };
```

Full Solution (exercise2.js)

```
function getFullName(person) {
  const { firstName, lastName } = person;
  return `${firstName} ${lastName}`;
}

module.exports = { getFullName };
```

Exercise 3: Build a Greeting with Template Literals

Problem Statement

In `exercise3.js`, implement `createGreeting` using ES6 template literals instead of string concatenation.

TODOs

- Change string concatenation to a template literal.
- Format: `"Hi <name>, welcome to <city>!"`

Starter Code (exercise3.js)

```
function createGreeting(name, city) {
  // TODO: Use template literal
  return "Hi " + name + ", welcome to " + city + "!";
}

module.exports = { createGreeting };
```

Full Solution (exercise3.js)

```
function createGreeting(name, city) {
  return `Hi ${name}, welcome to ${city}!`;
}

module.exports = { createGreeting };
```

Test Cases

To validate your solutions, create the following Jest test files:

exercise1.test.js

```
const { add, multiply, greet } = require("./exercise1");

test("add: sums two numbers", () => {
  expect(add(3, 7)).toBe(10);
  expect(add(-2, 5)).toBe(3);
});

test("multiply: multiplies two numbers", () => {
  expect(multiply(4, 5)).toBe(20);
  expect(multiply(0, 10)).toBe(0);
});

test("greet: returns greeting message", () => {
  expect(greet("Odessa")).toBe("Hello, Odessa!");
  expect(greet("Pinoy")).toBe("Hello, Pinoy!");
});
```

exercise2.test.js

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

test("getFullName: combines first and last name", () => {
  const person1 = { firstName: "Juan", lastName: "Dela Cruz" };
  expect(getFullName(person1)).toBe("Juan Dela Cruz");

  const person2 = { firstName: "Maria", lastName: "Clara" };
  expect(getFullName(person2)).toBe("Maria Clara");
});
```

exercise3.test.js

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

test("createGreeting: builds welcome message", () => {
  expect(createGreeting("Liza", "Cebu")).toBe("Hi Liza, welcome to Cebu!");
  expect(createGreeting("Andrei", "Davao")).toBe(
    "Hi Andrei, welcome to Davao!"
  );
});
```

Closing Story

As the clock struck 8 PM, the Quezon City sky was painted in shades of orange and violet. Odessa leaned back, sipping her iced matcha latte, feeling proud. She had successfully refactored code using arrow functions, learned how destructuring could make her life easier, and played with template literals for cleaner strings.

Her phone buzzed—a message from Kuya Ronnie:
“Great job, Ods! Next up: Promises and Async/Await. Ready to dance with asynchronous JavaScript?”

Odessa smiled. Every new ES6 feature felt like unlocking another level in her coding game. With these modern tools, she knew she could write cleaner, faster, and more maintainable code. Soon enough, she envisioned herself leading a startup, mentoring juniors, and inspiring other Pinoy coders to embrace the modern JavaScript mindset.

Steeling herself for the next challenge, she closed her laptop, excited for the promise of callbacks turned promises, and beyond. 🚀