🎬 Episode 18: Higher-Order Functions ft. Functional Programming

❓ What is a Higher-Order Function (HOF)?

A **Higher-Order Function** is a function that either:

- Takes another function as an argument, **or**
- Returns a function as its result.

📌 **Example**:

```
function x() {
    console.log("Hi");
}

function y(callback) {
    callback(); // Executes the function passed
}

y(x); // Output: Hi
```

> ☑ Here, y is a **Higher-Order Function**, and x is a **Callback Function**.

🧠 Problem Statement:

We have an array of radii, and we want to calculate:

1. 🟣 Area of each circle
2. 🟣 Circumference of each circle

🛠 First (Naive) Approach:

```
const radius = [1, 2, 3, 4];

const calculateArea = function(radius) {
    const output = [];
    for (let i = 0; i < radius.length; i++) {
        output.push(Math.PI * radius[i] * radius[i]);
    }
    return output;
};

console.log(calculateArea(radius));
```

> ☑ This works fine!
> ❌ But violates the **DRY Principle** (Don't Repeat Yourself) if we now want to calculate **circumference**.

🧪 Second Repetitive Approach:

```
const radius = [1, 2, 3, 4];

const calculateCircumference = function(radius) {
    const output = [];
    for (let i = 0; i < radius.length; i++) {
        output.push(2 * Math.PI * radius[i]);
    }
    return output;
};

console.log(calculateCircumference(radius));
```

> ❌ Still redundant – similar logic being reused with small changes.

☑ Refactored Functional Approach:

```
const radiusArr = [1, 2, 3, 4];

// Logic to calculate area
const area = function(radius) {
    return Math.PI * radius * radius;
};

// Logic to calculate circumference
const circumference = function(radius) {
    return 2 * Math.PI * radius;
};

// Higher-Order Function to apply any operation
const calculate = function(radiusArr, operation) {
    const output = [];
    for (let i = 0; i < radiusArr.length; i++) {
        output.push(operation(radiusArr[i]));
    }
    return output;
};

console.log(calculate(radiusArr, area));         // 👉 Calculates area
console.log(calculate(radiusArr, circumference)); // 👉 Calculates circumference
```

> 💧 Here, `calculate` is a **Higher-Order Function** that allows us to abstract out operations.
> 🧠 This is the **essence of Functional Programming** — reusability and clean abstraction.

🧪 Bonus: Polyfill of `Array.prototype.map()` 🧪

The `calculate` function above is a **custom implementation** of the native `.map()` method!

```
console.log(radiusArr.map(area)); // ☑ Native map
console.log(calculate(radiusArr, area)); // ☑ Our version (Polyfill)
```

🧩 Let's Build Our Own `.map()` Function (Polyfill Style):

```
Array.prototype.calculate = function(operation) {
    const output = [];
    for (let i = 0; i < this.length; i++) {
        output.push(operation(this[i]));
    }
    return output;
};

console.log(radiusArr.calculate(area)); // 👏 Works just like map()
```

💡 Takeaway:

- **Higher-Order Functions** are powerful tools for abstraction and code reuse.
- JavaScript embraces **Functional Programming** through features like **callbacks**, **HOFs**, and **methods like** `.map()`, `.filter()`, `.reduce()`.
- Always look to **refactor repetitive logic** into reusable HOFs for cleaner, maintainable code.
