🧠 JavaScript Deep Concepts – Practice & Theory

🔁 IIFE – Immediately Invoked Function Expression

```
// Runs as soon as it's defined
(function () {
  console.log("IIFE");
})();

// Use-case: Avoid polluting global scope or to immediately execute async
functions
let a = () => {
  return new Promise((resolve) => {
    setTimeout(() => resolve(456), 4000);
  });
};

(async () => {
  let b = await a();
  console.log(b);
  let c = await a();
  console.log(c);
  let d = await a();
  console.log(d);
})();

// console.log(d); // ❌ Error: d is not defined outside
```

🧩 Destructuring & Spread Operator

✅ Array Destructuring

```
let arr = [3, 5, 8, 9, 12, 14];
let [a, b, c, d, ...rest] = arr;
console.log(a, b, c, d, rest);
```

✅ Object Destructuring

```
let { a, b } = { a: 1, b: 5 };
console.log(a, b);
```

✅ Spread Syntax

```
let arr1 = [3, 5, 8];
let obj1 = { ...arr1 }; // Spreads keys as indices
console.log(obj1);

function sum(v1, v2, v3) {
  return v1 + v2 + v3;
}
console.log(sum(...arr1)); // 3 + 5 + 8

let obj2 = {
  name: "Harry",
  company: "Company xyz",
  address: "XYZ"
};
console.log({ ...obj2, name: "John" }); // John overrides Harry
console.log({ name: "John", ...obj2 }); // Harry overrides John
```

🌐 Scope in JavaScript

- `var` → Function/Global Scope
- `let` & `const` → Block Scope

```
let p = 9;
function ax() {
  let a = 8;
  console.log(p); // 9
  console.log(a); // 8
}
ax();
console.log(p); // 9
// console.log(a); // ❌ ReferenceError
```

🚀 Hoisting

```
console.log(a); // undefined due to hoisting
var a = 9;

// console.log(num); // ❌ ReferenceError
// let num = 6;
```

> Function declarations are hoisted, but function expressions and classes are **not**.

🔒 Closures

```
function init() {
  var name = "Mozilla";
  function displayName() {
    console.log(name);
  }
  name = "Harry";
  return displayName;
}
let c = init();
c(); // Harry
```

```
function returnFunc() {
  const x = () => {
    let a = 1;
    console.log(a);
    const y = () => {
      console.log(a);
      const z = () => {
        console.log(a);
      };
      z();
    };
    a = 999;
    y();
  };
  return x;
}
let a = returnFunc();
a();
```

> **Closure = Function + Lexical Environment**

🏹 Arrow Functions and `this`

```
const sayHello = (name) => {
  console.log("Greeting", name);
};

const x = {
  name: "Harry",
  role: "JS Developer",
  exp: 30,
  show: function () {
    setTimeout(() => {
      console.log(`The name is ${this.name}\nThe role is ${this.role}`);
    }, 2000);
  }
};

x.show(); // Uses lexical `this` inside arrow
```

🧪 Practice – Async & Spread

```
const a = async (text, n = 2) => {
  return new Promise((resolve) => {
    setTimeout(() => resolve(text), 1000 * n);
  });
};

(async () => {
  let text = await a("Hello");
  console.log(text);
  text = await a("World");
  console.log(text);
})();
```

```
function sum(a, b, c) {
  return a + b + c;
}
console.log(sum(...[1, 3, 5]));
```

💰 Simple Interest

```
function simpleInterest(p, r, t) {
  return (p * r * t) / 100;
}
console.log(simpleInterest(100, 5, 1)); // 5
```

📃 Regular Expressions (Regex)

```
const regex = /(Harry){2}/gi;
const text = "Harryharry is a very very nice awesome nice very boy";
console.log(text.replace(regex, "VERY"));
```

> Try Regex online: https://regexr.com

⌛ Event Loop & Asynchronous JavaScript

```
setTimeout(() => {
  console.log("You clicked the button!");
}, 2000);

console.log("Hi!");

setTimeout(() => {
  console.log("Click the button!");
}, 5000);

console.log("Welcome to loupe.");
```

> JS uses a **call stack**, **task queue**, and **Web APIs**. Use http://latentflip.com/loupe/ to visualize.

📦 Modules in JavaScript

☑ CommonJS (Node.js)

```
const hello = () => console.log("Hello Harry");
const ahello = (name) => console.log("Hello " + name);

module.exports = { hello, ahello };
```

☑ ES6 Modules

```
export const hello = () => console.log("Hello Harry");
export const ahello = (name) => console.log("Hello " + name");

const harry = () => console.log("Hello Harry");
export default harry;
```
