Mentorship

---

1. Introduction (5-10 mins)

- What is JavaScript? (Brief history)
  - A versatile, high-level programming language used in web development.
  - Runs in browsers, and now widely used on the server-side with Node.js.
- **Why ES6?**
  - Introduced in 2015 to solve JavaScript's growing complexity and provide modern syntax.
  - Makes code more readable, maintainable, and efficient.
  - Adds powerful new features like `let`, `const`, modules, and async programming.

Example

```javascript
// Before ES6
var name = "Pokhara";
function greet() {
  console.log("Hello " + name);
}
// ES6
const name = "Pokhara";
const greet = () => console.log(`Hello ${name}`);
```

---

2. JavaScript Essentials Refresher (15 mins)

1. **Variables**
   - `var` vs `let` vs `const`: Scope and immutability.
   - Best practice: Use `const` by default, `let` when reassignment is needed, avoid `var`.

   Example:

   ```javascript
   var x = 10; // Function-scoped
   let y = 20; // Block-scoped
   const z = 30; // Block-scoped, immutable
   if (true) {
       let y = 50; // New block-scoped variable
       console.log(y); // 50
   }
   console.log(y); // 20
   ```
2. **Basic DOM Manipulation**
   - Example of selecting elements and updating content:

JavaScript

```javascript
document.querySelector('#myButton').addEventListener('click', () => {
    document.querySelector('#message').textContent = "Button Clicked!";
});
```

---

3. ES6 Features (~60 mins)

A. Let & Const (5 mins)

1. `let` is block-scoped.
2. `const` is also block-scoped and immutable (value cannot be reassigned, though objects/arrays are still mutable).

Example to Show:

JavaScript

```javascript
let x = 5;
if (true) {
    let x = 10; // This x is different from the outer x
    console.log(x); // 10
}
console.log(x); // 5
const y = 20;
// y = 30; // Error
const obj = { name: 'Alice' };
obj.name = 'Bob'; // Allowed
console.log(obj.name); // Bob
```

B. Arrow Functions (10 mins)

1. Concise syntax.
2. Lexical `this` (inherits `this` from the surrounding scope).
3. No `arguments` object (use rest parameters instead).

Example:

JavaScript

```javascript
// Regular function
function sum(a, b) {
    return a + b;
}
sum()
// Arrow function
const sum = (a, b) => a + b;
// Arrow function and `this`
const person = {
    name: 'John',
    sayHi: () => {
        console.log(`Hi, I'm ${this.name}`); // 'this' does NOT refer to person
    }
};
person.sayHi();
// Because `arrow function` doesnot create its own `this`. It gets from where it is
defined i.e `person` and scope for `person` is global or undefined

// Correction
const person = {
    name: 'John',
    sayHi: function() {
          console.log(`Hi, I'm ${this.name}`); // 'this' refers to person
    }
};
person.sayHi();
```

C. Template Literals (5 mins)

1. String interpolation with `${}`.
2. Multi-line strings.

Example:

JavaScript

```javascript
const name = "John";
const age = 30;
console.log(`My name is ${name} and I am ${age} years old.`);
const multiLine = `This is line one.
This is line two.`;
console.log(multiLine);
```

D. Destructuring (10 mins)

1. Extract values from arrays/objects.
2. Assign default values.

Examples:

JavaScript

```javascript
// Array destructuring
const [a, b, c = 10] = [1, 2];
console.log(a, b, c); // 1, 2, 10
// Object destructuring
const user = { name: 'Alice', age: 25 };
const { name, age } = user;
console.log(name, age); // Alice, 25

// Renaming
const user = { firstName: 'Bob', lastName: 'Smith' };
const { firstName: fName, lastName: lName } = user;
console.log(fName, lName); // Output: Bob Smith

// Nested destructuring
const user = { name: 'Alice', address: { city: 'Wonderland', zip: '12345' } };
const { name, address: { city, zip } } = user;
console.log(name, city, zip); // Output: Alice Wonderland 12345
```

E. Default Parameters (5 mins)

1. Specify default values for function parameters.

Example:

JavaScript

```javascript
function greet(name = "Guest") {
    console.log(`Hello, ${name}!`);
}
greet(); // Hello, Guest!
greet("Alice"); // Hello, Alice!
greet(undefined); // Explicit undefined, so name = "Guest"
greet(null);      // Output: Hello, null! (default is NOT used because null is
explicitly passed)

// null is an actual value that represents "no value" or "empty."
// undefined means "no value was provided."
```

F. Rest and Spread Operators (10 mins)

1. **Rest:** Gather remaining arguments into an array.
2. **Spread:** Expand arrays/objects.

Example:

JavaScript

```javascript
// Rest
function sum(...numbers) {
    return numbers.reduce((total, num) => total + num, 0);
}
console.log(sum(1, 2, 3)); // 6
// Spread
const arr = [1, 2, 3];
const newArr = [...arr, 4, 5];
console.log(newArr); // [1, 2, 3, 4, 5]
```

G. Modules (10 mins)

1. `export` and `import` keywords for modular code.

Example: math.js:

JavaScript

```javascript
export const pi = 3.14159;
export function add(x, y) {
    return x + y;
}
export function multiply(x, y) {
    return x * y;
}

export default function greet(name) {
    console.log(`Hello, ${name}!`);
}

export {multiply}
export const add = (a, b) => a + b;
export const subtract = (a, b) => a - b;
```

main.js:

JavaScript

```javascript
import { add, subtract } from './math.js';
console.log(add(5, 3)); // 8
console.log(subtract(5, 3)); // 2
```

H. Promises and Async/Await (5-10 mins)

1. Introduce asynchronous programming.
2. Use promises for handling async tasks.
3. Simplify with `async/await`.

Examples:

JavaScript

```javascript
// Promise
const myPromise = new Promise((resolve, reject) => {
    // Simulate an asynchronous operation
    const success = true;  // Set this to 'false' to see the rejected state
    setTimeout(() => {
        if (success) {
            resolve("Operation completed successfully");
        } else {
            reject("Operation failed");
        }
    }, 2000);
});

myPromise
    .then((result) => {
        console.log("Success:", result);  // "Success: Operation completed 
successfully"
    })
    .catch((error) => {
        console.error("Error:", error);  // Will be called if the Promise is rejected
    });


const fetchData = () => {
    return new Promise((resolve, reject) => {
        setTimeout(() => resolve("Data loaded"), 1000);
    });
};
fetchData().then(data => console.log(data));

// Async/Await
async function fetchData() {
    return new Promise((resolve) => {
        setTimeout(() => resolve("Data fetched"), 1000);
    });
}

async function main() {
    console.log("Fetching data...");
    const result = await fetchData();  // Wait until fetchData() is resolved
    console.log(result);  // "Data fetched"
}

main();

const loadData = async () => {
    const data = await fetchData();
    console.log(data);
};
loadData();
```

4. Practice/Interactive Examples (15-20 mins)

Small Exercises:

1. Swap two variables using destructuring:

```javascript
let a = 1, b = 2;
[a, b] = [b, a];
console.log(a, b); // 2, 1
```

2. Write a function with default parameters:

```javascript
const greet = (name = "Guest") => `Hello, ${name}!`;
console.log(greet()); // Hello, Guest!
console.log(greet("Alice")); // Hello, Alice!
```

3. Use `fetch` with async/await:

```javascript
const fetchData = async () => {
    const response = await fetch('https://api.example.com/data');
    const data = await response.json();
    console.log(data);
};
fetchData();
```

---

5. Q&A and Recap (5-10 mins)
