JavaScript Assignment – Day 3

Date: 10th July 2025
Organisation: Kreeti Technologies
Bootcamp: React and AI BootCamp

Assignment Submitted By:
Alok Agarwal

---

1. Lexical Scoping

Lexical scoping means that a variable is accessible only within the block where it was defined, and in any blocks nested inside it. It cannot be accessed from outside that scope.

Example:

```
function outer() {
  let name = "Alok";
  function inner() {
    console.log(name); // Can access 'name' from outer function
  }

  inner();
}
outer();
```

Trying to access the variable name outside the outer() function would result in an error.

---

2. Higher-Order Functions

A higher-order function is one that does either of the following:

- Takes another function as an argument (like a callback).
- Returns another function as a result.

Examples:

```
function greet(name) {
  return "Hello, " + name;
}
function saySomething(fn, value) {
  console.log(fn(value)); // Calling 'greet' inside 'saySomething'
}
saySomething(greet, "Alok");
```

Another example using a function that returns a function:

```
function multiplier(x) {
  return function(y) {
    return x * y;
  };
}
const double = multiplier(2);
console.log(double(5)); // Outputs: 10
```

---

3. some() and every() Array Methods

- some() checks if at least one item in the array passes a test.
- every() checks if all items in the array pass a test.

Example:

```
const nums = [2, 4, 6];
console.log(nums.some(n => n % 2 !== 0)); // false
console.log(nums.every(n => n % 2 === 0)); // true
```

---

4. Behavior of this in Normal vs Arrow Functions

- In a normal function, this refers to the object that called the function.
- In an arrow function, this is inherited from the surrounding (lexical) context and not bound to the object.

Example:

```
const person = {

  name: "Alok",

  normalFunc: function() {

    console.log(this.name); // Outputs: Alok

  },

  arrowFunc: () => {

    console.log(this.name); // Outputs: undefined

  }

};

person.normalFunc();

person.arrowFunc();
```

Can arrow functions be used as object methods?
Not ideally. Arrow functions don’t have their own this, so using them as object methods can lead to unexpected behavior.

Practice Questions

Q1. Destructuring

Given the object:

```
const myObject = {
  x1: "Samba",
  x2: {
    x3: {
      x4: {},
      x5: "Rails"
    },
    x6: {
      x7: -1,
      x8: [25, 8, 4, 10]
    }
  }
};
```

a. Get x3 in a variable y

```
const { x2: { x3: y } } = myObject;
console.log(y);
```

b. Get the 2nd element of x8

```
const { x2: { x6: { x8: [, secondValue] } } } = myObject;
console.log(secondValue); // Output: 8
```

Q2. Create newObject by copying the old one and adding values

```
const newObject = {
  ...myObject,
  newKey1: "value1",
  newKey2: "value2"
};

console.log(newObject);
```

Q3. Class Vehicle and Subclass Car

```
class Vehicle {
  constructor(brand, speed) {
    this.brand = brand;
    this.speed = speed;
  }

  move() {
    console.log(`${this.brand} is moving at ${this.speed} km/h`);
  }
}
```

```
class Car extends Vehicle {
  constructor(brand, speed, fuelType) {
    super(brand, speed); // Call parent constructor
    this.fuelType = fuelType;
  }
  refuel() {
    console.log(`${this.brand} is refueling with ${this.fuelType}`);
  }
}
const myCar = new Car("Toyota", 80, "Petrol");
myCar.move();      // Output: Toyota is moving at 80 km/h
myCar.refuel();    // Output: Toyota is refueling with Petrol
```

Q4. Array Methods with First 10 Numbers

```
const arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
```

a. Print even numbers

```
const evens = arr.filter(n => n % 2 === 0);
console.log(evens); // Output: [2, 4, 6, 8, 10]
```

b. Sum all numbers using .reduce()

```
const total = arr.reduce((acc, curr) => acc + curr, 0);
console.log(total); // Output: 55
```
