Assignment 2:

Name: Dipanshu
Student ID: GF202345540
Subject: Node & ReactJS Development
Semester: 5TH

Repo link:
https://github.com/Dipanshudk46/nodejs-assignment-2

Task 1: Hoisting in Variables

Write a Node.js program that demonstrates variable hoisting using var, let, and const.
Print a variable before it is declared.
Show the difference between var, let, and const.
Explain the output.

hosting_variable.js > ...

```
// Using var
console.log("var before declaration:", myVar);
var myVar = 10;
console.log("var after declaration:", myVar);

// Using let
try {
    console.log("let before declaration:", myLet);
} catch (error) {
    console.log("Error with let before declaration:", error.message);
}
let myLet = 20;
console.log("let after declaration:", myLet);

// Using const
try {
    console.log("const before declaration:", myConst);
} catch (error) {
    console.log("Error with const before declaration:", error.message);
}
const myConst = 30;
console.log("const after declaration:", myConst);
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE PORTS

```
PS D:\node js assignment\Assignment 2> node "d:\node js assignment\Assignment 2\hosting_variable.js"
var before declaration: undefined
var after declaration: 10
Error with let before declaration: Cannot access 'myLet' before initialization
let after declaration: 20
Error with const before declaration: Cannot access 'myConst' before initialization
const after declaration: 30
PS D:\node js assignment\Assignment 2>
```

Explanation:

- **var** is hoisted and initialized with undefined.
- **let** and **const** are hoisted but stay in the **temporal dead zone** until declared → calling them early causes an error.
- **const** must also be assigned a value at declaration.

Task 2: Function Declarations vs Expressions

Create two functions in Node.js:
A function declaration (function add(a,b) {})
A function expression (const multiply = function(a,b) {})
Call both functions before and after their definitions.
Record what works and what fails.
Explain why.

function dec.js > ...

```
1   // Function Declaration
2   console.log("Calling add() before declaration:", add(5, 10));
3
4   function add(a, b) {
5       return a + b;
6   }
7
8   console.log("Calling add() after declaration:", add(15, 20));
9
10  // Function Expression
11  try {
12      console.log("Calling multiply() before declaration:", multiply(2, 3));
13  } catch (error) {
14      console.log("Error when calling multiply() before declaration:", error.message);
15  }
16
17  const multiply = function(a, b) {
18      return a * b;
19  };
20
21  console.log("Calling multiply() after declaration:", multiply(4, 5));
22
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE PORTS

```
PS D:\node js assignment\Assignment 2> node "d:\node js assignment\Assignment 2\function dec.js"
Calling add() before declaration: 15
Calling add() after declaration: 35
Error when calling multiply() before declaration: Cannot access 'multiply' before initialization
Calling multiply() after declaration: 20
PS D:\node js assignment\Assignment 2>
```

Explanation:

Function Declarations are hoisted → can be used before or after definition.
Function Expressions (with const) are not hoisted the same way → cannot be used before initialization.

Task 3: Arrow Functions vs Normal Functions

Create two functions inside an object:
One arrow function
One normal function
Both should print this.
Compare their outputs when called as methods of the object.

arrow_normal.js > ...

```
1   const obj = {
2       name: "NodeJS",
3       normalFunc: function() {
4           console.log("Normal Function this.name:", this.name);
5       },
6       arrowFunc: () => {
7           console.log("Arrow Function this.name:", this.name);
8       }
9   };
10
11  obj.normalFunc();
12  obj.arrowFunc();
13
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE PORTS

```
PS D:\node js assignment\Assignment 2> node "d:\node js assignment\Assignment 2\arrow_normal.js"
Normal Function this.name: NodeJS
Arrow Function this.name: undefined
PS D:\node js assignment\Assignment 2>
```

Explanation:

- **Normal functions** bind this to the object calling them → obj.normalFunc() prints "NodeJS".
- **Arrow functions** do not have their own this; they use the surrounding scope's this → in this case, not the object, so it prints undefined.

Task 4: Higher Order Functions

Write a Node.js function calculate(operation, a, b) where operation is another function (like add, subtract).
Pass different functions to calculate and print results.
*Example: calculate((x,y) => x*y, 4, 5) should return 20.*

high_order.js > ...

```
1   function calculate(operation, a, b) {
2       return operation(a, b);
3   }
4
5   // Different operations
6   const add = (x, y) => x + y;
7   const subtract = (x, y) => x - y;
8   const multiply = (x, y) => x * y;
9   const divide = (x, y) => x / y;
10
11  // Using calculate with different functions
12  console.log("Add:", calculate(add, 10, 5));
13  console.log("Subtract:", calculate(subtract, 10, 5));
14  console.log("Multiply:", calculate(multiply, 4, 5));
15  console.log("Divide:", calculate(divide, 20, 4));
16  console.log("Power:", calculate((x, y) => x ** y, 2, 3)); // Inline arrow function for power
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE PORTS

```
PS D:\node js assignment\Assignment 2> node "d:\node js assignment\Assignment 2\high_order.js"
Add: 15
Subtract: 5
Multiply: 20
Divide: 5
PS D:\node js assignment\Assignment 2>
```

Explanation:

- **Higher Order Functions** can take other functions as arguments.
- Here, calculate accepts an operation function and applies it to a and b.
- This makes the function flexible for **reusability and modularity**.