Assignment – 6

Section 1

1. Shopping Cart Application

```
const taxRate = 0.07; // constant for tax rate
let items = [];

function addItem(name, price) {
   items.push({ name, price });
}

function calculateTotal() {
   let total = items.reduce((sum, item) => sum + item.price, 0);
   return total + (total * taxRate);
}

// Example usage
addItem("Book", 15);
addItem("Pen", 2);
console.log("Total Price:", calculateTotal());
```

2. Area of a Rectangle

```
// Regular function
function calculateArea(length, width) {
   return length * width;
}

// Arrow function
const calculateAreaArrow = (length, width) => length * width;

// Example usage
console.log(calculateArea(5, 10)); // 50
console.log(calculateAreaArrow(5, 10)); // 50
```

3. Book Object

```
const book = {
   title: "1984",
   author: "George Orwell",
   year: 1949,
   displayDetails() {
      console.log(`${this.title} by ${this.author}, published in ${this.year}`);
   }
};

// Example usage
book.displayDetails(); // 1984 by George Orwell, published in 1949
```

4. Object Destructuring

```
const car = {
   make: "Toyota",
   model: "Camry",
   year: 2021
};

const { make, model, year } = car;
console.log(make, model, year); // Toyota Camry 2021
```

5. Array Destructuring

```
const numbers = [10, 20, 30, 40];
const [first, second] = numbers;
console.log(first, second); // 10 20
```

6. Map Method

```
const names = ["Alice", "Bob", "Charlie"];
const nameLengths = names.map(name => name.length);
console.log(nameLengths); // [5, 3, 7]
```

7. Filter Method

```
const numbersArray = [1, 2, 3, 4, 5, 6];
const evenNumbers = numbersArray.filter(num => num % 2 === 0);
```

```
console.log(evenNumbers); // [2, 4, 6]
```

8. Reduce Method

```
const cartItems = [
   { name: "Book", price: 15 },
   { name: "Pen", price: 2 },
   { name: "Notebook", price: 5 }
];

const totalPrice = cartItems.reduce((total, item) => total + item.price, 0);
console.log(totalPrice); // 22
```

9. Function with Rest Operator

```
function sum(...args) {
   return args.reduce((total, num) => total + num, 0);
}

// Example usage
console.log(sum(1, 2, 3, 4)); // 10
```

10. Spread Operator

```
const fruits1 = ["apple", "banana"];
const fruits2 = ["orange", "grape"];
const allFruits = [...fruits1, ...fruits2];
console.log(allFruits); // ["apple", "banana", "orange", "grape"]
```

11. Function with Delay

```
function executeAfterDelay(callback, delay) {
   setTimeout(callback, delay);
}

// Example usage
executeAfterDelay(() => console.log("Executed after delay!"), 2000);
```

12. Promise

```
const myPromise = new Promise((resolve) => {
```

```
    setTimeout(() => {
        resolve("Promise resolved after 3 seconds");
    }, 3000);
});

// Example usage
myPromise.then(message => console.log(message));
```

13. Closure Demonstration

```
function outerFunction() {
    let count = 0;
    return function innerFunction() {
        count++;
        console.log(count);
    };
}

const counter = outerFunction();
counter(); // 1
counter(); // 2
```

14. Async/Await with Fetch

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

// Example usage
fetchData();
```

15. Filter, Map, and Reduce

```
function processNumbers(numbers) {
    return numbers
        .filter(num => num % 2 === 0) // Keep even numbers
        .map(num => num * 2)          // Double those numbers
        .reduce((total, num) => total + num, 0); // Total them up
}
```

```
// Example usage
const nums = [1, 2, 3, 4, 5, 6];
console.log(processNumbers(nums)); // 24 (2*2 + 4*2 + 6*2)
```

SECTION 2

Description: Create simple Personal Budget Tracker application that allows users to manage their expenses. The application should include functionalities to add, view, and calculate the total expenses. You will utilize various JavaScript concepts to implement this application.

Requirements:

1. Variables: Use let, const, and var to manage state variables like expense list and total expense.

```
// Personal Budget Tracker

// 1. State Variables
const initialExpenses = [
    { description: "Groceries", amount: 50, date: "2024-10-01" },
    { description: "Utilities", amount: 30, date: "2024-10-02" },
];

let expenses = [...initialExpenses]; // Spread operator to initialize expenses
let totalExpense = 0; // Variable to hold the total expense
```

2. Functions and Arrow Functions: Create functions to add an expense, display all expenses, and calculate the total. Use an arrow function for at least one of these.

```
// 2. Functions
const addExpense = (description, amount, date) => {
```

```
    const newExpense = { description, amount, date };
    expenses = [...expenses, newExpense]; // Spread operator to create new expense list
    updateTotal();
    displaySuccessMessage(() => console.log(`Added: ${description}`));
};

function displayExpenses() {
    expenses.forEach(({ description, amount, date }) => {
        console.log(`Description: ${description}, Amount: $${amount}, Date: ${date}`);
    });
}
```

3. JavaScript Objects:Represent each expense as an object with properties such as description, amount, and date

```
// 3. Calculate total expenses
function updateTotal() {
    totalExpense = expenses.reduce((total, expense) => total + expense.amount, 0);
}
```

4. Destructuring: Use array and object destructuring when retrieving expense details for display.

```
// 4. Displaying expenses
const displayExpenseDescriptions = () => {
    const descriptions = expenses.map(({ description }) => description);
    console.log("Expense Descriptions:", descriptions);
};
```

5. Array Methods (Map, Filter, Reduce):

○ Use map to display a list of expense descriptions.

○ Use filter to show only expenses above a certain amount (e.g., $20).

○ Use reduce to calculate the total expenses.

```
// 5. Filter expenses above a certain amount
const filterExpensiveItems = (minAmount) => {
```

```
    const filteredExpenses = expenses.filter(({ amount }) => amount > minAmount);
    console.log(`Expenses above $${minAmount}:`, filteredExpenses);
};
```

6. Rest and Spread Operator: Use the rest operator to allow adding multiple expenses at once. Use the spread operator to create a new expense list when adding new expenses.

```
// 6. Rest Operator for adding multiple expenses
function addMultipleExpenses(...newExpenses) {
    newExpenses.forEach(({ description, amount, date }) => addExpense(description, amount, date));
}
```

7. Callback Functions: Implement a function that takes a callback to display a success message after an expense is added.

```
// 7. Callback Function for success message
function displaySuccessMessage(callback) {
    callback();
}
```

8. Promises: Create a promise that simulates fetching initial expenses from an API (you can just resolve with a hard-coded array).

```
// 8. Promise to simulate fetching initial expenses
function fetchInitialExpenses() {
    return new Promise((resolve) => {
        setTimeout(() => {
            resolve(initialExpenses);
        }, 1000);
    });
}
```

9. Closures: Use a closure to create a function that maintains the state of total expenses.

```
// 9. Closure for maintaining total expense state
function createTotalExpenseTracker() {
    let total = 0;

    return function updateTotal(expense) {
```

```
        total += expense.amount;
        console.log("Current Total Expense: $" + total);
    };
}

const totalTracker = createTotalExpenseTracker();
```

10. Async/Await: Use asyncc/await to fetch initial expenses and display them in the application when it load

```
// 10. Async/Await to fetch and display initial expenses
async function initializeApp() {
    const fetchedExpenses = await fetchInitialExpenses();
    expenses = [...fetchedExpenses]; // Spread operator to set expenses
    expenses.forEach(expense => totalTracker(expense)); // Update total for each fetched expense
    displayExpenses();
    displayExpenseDescriptions();
    filterExpensiveItems(20);
}

// Initialize the application
initializeApp();

// Example usage of adding new expenses
addMultipleExpenses(
    { description: "Restaurant", amount: 25, date: "2024-10-05" },
    { description: "Fuel", amount: 40, date: "2024-10-06" }
);
```

Explanation:

1. **State Variables**: initialExpenses and expenses to manage the expense list. The total expense is maintained in totalExpense.
2. **Functions and Arrow Functions**: Functions to add expenses and display them; an arrow function is used for addExpense.
3. **JavaScript Objects**: Each expense is an object with description, amount, and date properties.
4. **Destructuring**: Used in the displayExpenses and displayExpenseDescriptions functions.
5. **Array Methods**:
    - map for displaying descriptions.
    - filter to show expenses above $20.
    - reduce for calculating the total expenses.
6. **Rest and Spread Operators**: Used to add multiple expenses and to create a new expense list.
7. **Callback Functions**: Implemented in displaySuccessMessage.
8. **Promises**: Simulates fetching expenses with a promise.
9. **Closures**: The createTotalExpenseTracker function maintains the total expense state.
10. **Async/Await**: Used in initializeApp to fetch initial expenses.