Государственный Университет Молдовы

Факультет Математики и Информатики

Департамент Информатики

Лабораторная работа №2

Основы работы с массивами, функциями и объектами в JavaScript

Подготовила студентка группы IA2403:

Горбатюк Эвелина

Кишинэу 2025

1. Изучить основы работы с массивами и функциями в JavaScript, применяя их для обработки и анализа транзакций.
2. Создание массива транзакций: разработать массив объектов с необходимыми свойствами для анализа.
3. Реализация функций для анализа транзакций: создать функции для получения уникальных типов, подсчета сумм, фильтрации по критериям и других операций.
4. Тестирование функций: проверить работоспособность на различных наборах данных, включая пустые массивы.
5. Документирование кода: обеспечить понятное документирование с использованием JSDoc.
6. Ответы на контрольные вопросы: проанализировать методы работы с массивами в JavaScript.

Практическая часть: Консольное приложение для анализа транзакций

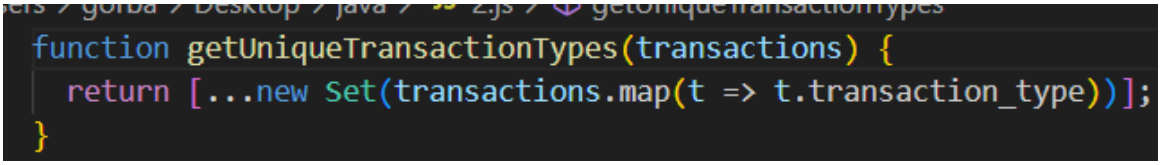
Цель работы

Разработка консольного приложения для анализа транзакций, которое позволяет выполнять различные операции с данными о транзакциях.

Описание приложения

Приложение реализует следующие функции:

1. **Получение уникальных типов транзакций**: возвращает массив уникальных типов транзакций.



2. **Расчет общей суммы транзакций**: вычисляет общую сумму всех транзакций.

```
function calculateTotalAmount(transactions) {
  return transactions.reduce((sum, t) => sum + t.transaction_amount, 0);
}
```

3. **Расчет суммы транзакций по дате**: вычисляет сумму транзакций за указанный день, месяц и год.

```
function calculateTotalAmountByDate(transactions, year, month, day) {
  return transactions.filter(t => {
      const date = new Date(t.transaction_date);
      return (!year || date.getFullYear() === year) &&
             (!month || date.getMonth() + 1 === month) &&
             (!day || date.getDate() === day);
  }).reduce((sum, t) => sum + t.transaction_amount, 0);
}
```

4. **Получение транзакций по типу**: фильтрует транзакции по указанному типу.

```
function getTransactionByType(transactions, type) {
  return transactions.filter(t => t.transaction_type === type);
}
```

5. **Получение транзакций в диапазоне дат**: фильтрует транзакции по заданному диапазону дат.

```
function getTransactionsInDateRange(transactions, startDate, endDate) {
  return transactions.filter(t => {
      const date = new Date(t.transaction_date);
      return date >= new Date(startDate) && date <= new Date(endDate);
  });
}
```

6. **Получение транзакций по торговцу**: фильтрует транзакции по названию торговца.

```
function getTransactionsByMerchant(transactions, merchantName) {
  return transactions.filter(t => t.merchant_name === merchantName);
}
```

7. **Расчет средней суммы транзакций**: вычисляет среднюю сумму транзакций.

```
function calculateAverageTransactionAmount(transactions) {
  return transactions.length ? calculateTotalAmount(transactions) / transactions.length : 0;
}
```

8. **Фильтрация транзакций по диапазону сумм**: возвращает транзакции в указанном диапазоне сумм.

```
function getTransactionsByAmountRange(transactions, minAmount, maxAmount) {
  return transactions.filter(t => t.transaction_amount >= minAmount && t.transaction_amount <= maxAmount);
}
```

9. **Расчет общей суммы дебетовых транзакций**: вычисляет сумму всех дебетовых транзакций.

```
function calculateTotalDebitAmount(transactions) {
  return calculateTotalAmount(getTransactionByType(transactions, "debit"));
}
```

10. **Поиск месяца с наибольшим количеством транзакций**: определяет месяц с наибольшим числом транзакций.

```
function findMostTransactionsMonth(transactions) {
  const count = {};
  transactions.forEach(t => {
      const month = new Date(t.transaction_date).getMonth() + 1;
      count[month] = (count[month] || 0) + 1;
  });
  return Object.keys(count).reduce((a, b) => count[a] > count[b] ? a : b);
}
```

11. **Поиск месяца с наибольшими дебетовыми транзакциями**: определяет месяц с наибольшими дебетовыми транзакциями.

```
function findMostDebitTransactionMonth(transactions) {
  return findMostTransactionsMonth(getTransactionByType(transactions, "debit"));
}
```

12. **Определение более частого типа транзакций**: сравнивает количество дебетовых и кредитовых транзакций.

```
function mostTransactionTypes(transactions) {
  const debitCount = getTransactionByType(transactions, "debit").length;
  const creditCount = getTransactionByType(transactions, "credit").length;
  return debitCount > creditCount ? "debit" : creditCount > debitCount ? "credit" : "equal";
}
```

13. **Получение транзакций до указанной даты**: возвращает транзакции, осуществленные до заданной даты.

```
function getTransactionsBeforeDate(transactions, date) {
  return transactions.filter(t => new Date(t.transaction_date) < new Date(date));
}
```

14. **Поиск транзакции по идентификатору**: находит транзакцию по ее идентификатору.

```
function findTransactionById(transactions, id) {
  return transactions.find(t => t.transaction_id === id) || null;
}
```

15. **Получение описаний транзакций**: возвращает массив описаний всех транзакций.

```
function mapTransactionDescriptions(transactions) {
  return transactions.map(t => t.transaction_description);
}
```

Тестирование

Функции тестируются с использованием заранее подготовленного массива тестовых транзакций.

```
// Проверка всех функций
console.log("Unique Transaction Types:", getUniqueTransactionTypes(testTransactions));
console.log("Total Amount:", calculateTotalAmount(testTransactions));
console.log("Total Amount by Date (2025-02-01):", calculateTotalAmountByDate(testTransactions, 2025, 2, 1));
console.log("Transactions by Type (debit):", getTransactionByType(testTransactions, "debit"));
console.log("Transactions in Date Range (Feb 2025 - Mar 2025):", getTransactionsInDateRange(testTransactions, "2025-02-01", "2025-03-31"));
console.log("Transactions by Merchant (Store1):", getTransactionsByMerchant(testTransactions, "Store1"));
console.log("Average Transaction Amount:", calculateAverageTransactionAmount(testTransactions));
console.log("Transactions by Amount Range (100-1000):", getTransactionsByAmountRange(testTransactions, 100, 1000));
console.log("Total Debit Amount:", calculateTotalDebitAmount(testTransactions));
console.log("Most Transactions Month:", findMostTransactionsMonth(testTransactions));
console.log("Most Debit Transaction Month:", findMostDebitTransactionMonth(testTransactions));
console.log("Most Transaction Types:", mostTransactionTypes(testTransactions));
console.log("Transactions Before Date (2025-02-02):", getTransactionsBeforeDate(testTransactions, "2025-02-02"));
console.log("Find Transaction by ID (1):", findTransactionById(testTransactions, "1"));
console.log("Transaction Descriptions:", mapTransactionDescriptions(testTransactions));
```

Заключение

Разработанное приложение позволяет эффективно анализировать транзакции, предоставляя пользователю возможность выполнять различные операции с данными.

1. Какие методы массивов можно использовать для обработки объектов в JavaScript?

- **`map()`**: для преобразования массива.
- **`filter()`**: для отбора элементов по условию.
- **`reduce()`**: для агрегирования значений в одно.
- **`forEach()`**: для перебора элементов без создания нового массива.

- **`find()`**: для поиска первого подходящего элемента.
- **`some()`**: для проверки наличия хотя бы одного подходящего элемента.
- **`every()`**: для проверки, соответствуют ли все элементы условию.

2. Как сравнивать даты в строковом формате в JavaScript?

```
const date1 = new Date("2025-02-01");
const date2 = new Date("2025-03-01");
const isEarlier = date1 < date2;
```

3. В чем разница между `map()`, `filter()` и `reduce()` при работе с массивами объектов?

- **`map()`**: создает новый массив, применяя функцию к каждому элементу.
- **`filter()`**: создает новый массив, содержащий только элементы, которые удовлетворяют условию.
- **`reduce()`**: агрегирует элементы массива в одно значение, используя функцию.