Молдавский Государственный Университет
Факультет Математики и Информатики
Департамент Информатики

Отчет по лабораторной работе по дисциплине „JavaScript”

Выполнила: студентка группы **IAFR2403 R**
Alexandra Ivarovscaia
Проверил преподаватель:
Nartea Nichita, lector univ.

Кишинев, 2025

Индивидуальная (лабораторная) работа №1

Цель: ознакомиться с основными функциями и с синтаксисом JavaScript на основе консольного приложения для анализа транзакций.

Условие: создание консольного приложения для анализа транзакций.

Структура проекта:

1. файл transactions.json, содержащий указанные транзакции
2. файл index.js для размещения основного кода

Выполнение работы:

Каждая транзакция представлена отдельным объектом, содержащим все необходимые данные. В каждую транзакцию добавлен метод string(), который возвращает строковое представление транзакции в формате JSON.

```
/**
 * @description class-template for transaction object
 * @constructor
 * @param {string} transaction_id - transaction id
 * @param {string} transaction_date - transaction date
 * @param {number} transaction_amount - transaction amount
 * @param {string} transaction_type - transaction type
 * @param {string} transaction_description - transaction description
 * @param {string} merchant_name - merchant name
 * @param {string} card_type - card type
 * @returns {string} transactionAsString - transaction in string format
 */
class Transaction {
  constructor({
    transaction_id,
    transaction_date,
    transaction_amount,
    transaction_type,
    transaction_description,
    merchant_name,
    card_type
  }) {
    this.transaction_id = transaction_id;
    this.transaction_date = transaction_date;
    this.transaction_amount = transaction_amount;
    this.transaction_type = transaction_type;
    this.transaction_description = transaction_description;
    this.merchant_name = merchant_name;
    this.card_type = card_type;
```

Рисунок 1: класс-конструктор объекта транзакций

```
    this.transaction_description = transaction_description;
    this.merchant_name = merchant_name;
    this.card_type = card_type;
  }
  string() {
    let transactionAsString = JSON.stringify(this);
    return transactionAsString;
  }
}
```

Рисунок 2: класс-конструктор объекта транзакций (продолжение)

```
/**
 * @description import array of transaction objects from JSON file
 */
const transactionsArray = require('C:/Users/37379/OneDrive/Desktop/University/JS/LI1/transaction.json');
```

Рисунок 3: массив объектов транзакций из файла

Класс TransactionAnalyzer для обработки транзакций. Конструктор класса принимает все транзакции в качестве аргумента. Добавлены методы для добавления новой транзакции addTransaction() и получения списка всех транзакций getAllTransaction().

```
/**
 * @description transaction analyzer class
 */
class TransactionAnalyzer {

  /**
   * @description transaction analyzer constructor
   * @constructor
   * @param {Array<Object>} transactions - transaction objects array
   */
   constructor(transactions) {
     this.transactions = transactions.map(transaction => new Transaction(transaction));
   }

  /**
   * @description add new transction method
   * @param {Object} newTransaction - new transaction object
   */
   addTransaction(newTransaction) {
     this.transactions.push(new Transaction(newTransaction));
   }

  /**
   * @descriiption get all transactions method
   * @returns {string} allTransactionsString - all transactions in string format
   */
   getAllTransaction() {
     let allTransactionsString = this.transactions.map(transaction => transaction.string()).join('\n');
     return allTransactionsString;
   }
```

Рисунок 4: класс анализа транзакций с методами добавления и получения транзакций

```
/**
 * @description TransactionAnalyzer class instance with transactionsArray as parameter
 */
const transactionsAnalysis = new TransactionAnalyzer(transactionsArray);
```

Рисунок 5: создание экземпляра класса анализатора транзакций

Реализованы методы для обработки данных о транзакциях:

1. Метод getUniqueTransactionType() - возвращает массив всевозможных типов транзакций

```
/**
 * @description get distinct transaction types array method
 * @returns {Array<string>} uniqueTypeTransactions - distinct transaction types in array
 */
 getUniqueTransactionType(){
   const allTypeTransactions = this.transactions.map(transaction => transaction.transaction_type);
   const uniqueTypeTransactions = [...new Set(allTypeTransactions)];
   return uniqueTypeTransactions;
 }
```

Рисунок 6: метод получения всех типов транзакций

2. Метод calculateTotalAmount() - рассчитывает общую сумму всех транзакций

```
/**
 * @description calculate total trasaction amount method
 * @returns {number} totalTransactionAmount - total amount of all transactions as number
 */
 calculateTotalAmount(){
   let totalTransactionAmount = this.transactions.reduce((total, transaction) => total + Number(transaction.transaction_amount), 0);
   return totalTransactionAmount;
 }
```

Рисунок 7: метод рассчёта суммы всех транзакций

3. Метод calculateTotalAmountByDate(year, month, day) - вычисляет общую сумму транзакций за указанный год, месяц и день

```
/**
 * @description calculate total trasaction amount by given year/month/day method
 * @param {number} year - transaction year
 * @param {number} month - transaction moth
 * @param {number} day - transaction day
 * @returns {number} totalTransactionAmountByDate - total transaction amount by date as number
 */
 calculateTotalAmountByDate(year, month, day){
   const FilteredTransactions = this.transactions.filter(transaction => {
     const transactionDate = new Date(transaction.transaction_date);
     const yearMatches = year ? transactionDate.getFullYear() === year : true;
     const monthMatches = month ? transactionDate.getMonth() + 1 === month : true;
     const dayMatches = day ? transactionDate.getDate() === day : true;
     return yearMatches && monthMatches && dayMatches;
   });
   let totalTransactionAmountByDate = FilteredTransactions.reduce((total, transaction) => total + Number(transaction.transaction_amount), 0);
   return totalTransactionAmountByDate;
 }
```

Рисунок 8: метод рассчёта суммы транзакций за указанный период времени

4. Метод getTransactionByType(type) - возвращает транзакции указанного типа (debit или credit)

```
/**
 * @description get transaction by type method
 * @param {string} type - transaction type
 * @returns {string} transactionByTypeString - transactions by type as string
 */
 getTransactionByType(type){
   const transactionsByType = this.transactions.filter(transaction => transaction.transaction_type === type);
   let transactionByTypeString = transactionsByType.map(transaction => transaction.string()).join('\n');
   return transactionByTypeString;
 }
```

Рисунок 9: метод получения транзакций определённого типа

5. Метод getTransactionsInDateRange(startDate, endDate) - возвращает транзакции, проведенные в указанном диапазоне дат от startDate до endDate

```
/**
 * @description get transactions in given date range method
 * @param {string} startDate - start date
 * @param {string} endDate - end date
 * @returns {string} transactionsInDateRange - transactions in date range as string
 */
 getTransactionsInDateRange(startDate, endDate){
   const start = new Date(startDate);
   const end = new Date(endDate);
   const filteredTransactions = this.transactions.filter(transaction => {
     const transactionDate = new Date(transaction.transaction_date);
     return transactionDate >= start && transactionDate <= end;
   });
   let transactionsInDateRange = filteredTransactions.map(transaction => transaction.string()).join('\n');
   return transactionsInDateRange;
 }
```

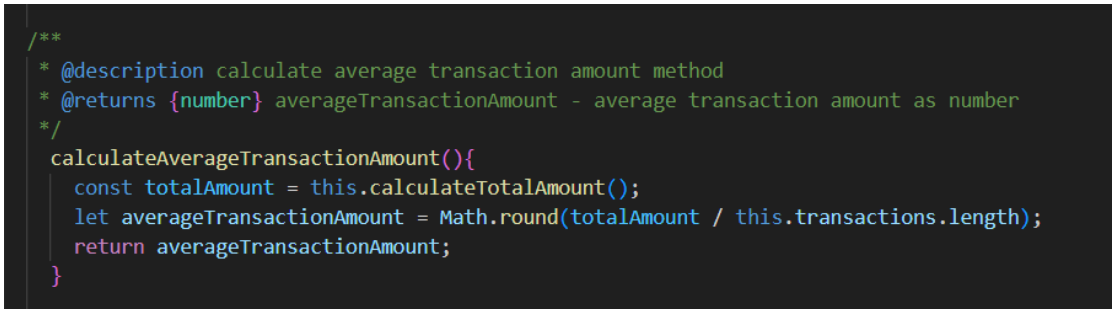
Рисунок 10: метод получения транзакций за указанный период времени

6. Метод getTransactionsByMerchant(merchantName) - возвращает транзакции, совершенные с указанным торговым местом или компанией

```
/**
 * @description get transaction by merchant method
 * @param {string} merchantName - merchant name
 * @returns {string} transactionByMerchantString - transactions by merchant as string
 */
 getTransactionsByMerchant(merchantName){
   const transactionByMerchant = this.transactions.filter(transaction => transaction.merchant_name === merchantName);
   let transactionByMerchantString = transactionByMerchant.map(transaction => transaction.string()).join('\n');
   return transactionByMerchantString;
 }
```

Рисунок 11: метод получения транзакций определённой компании

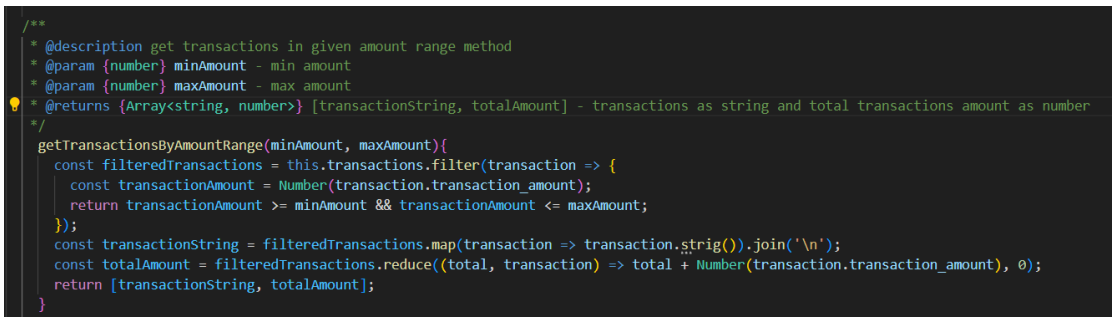
7. Метод calculateAverageTransactionAmount() - возвращает среднее значение транзакций



```
/**
 * @description calculate average transaction amount method
 * @returns {number} averageTransactionAmount - average transaction amount as number
 */
 calculateAverageTransactionAmount(){
   const totalAmount = this.calculateTotalAmount();
   let averageTransactionAmount = Math.round(totalAmount / this.transactions.length);
   return averageTransactionAmount;
 }
```

Рисунок 12: метод рассчёта среднего значения транзакций

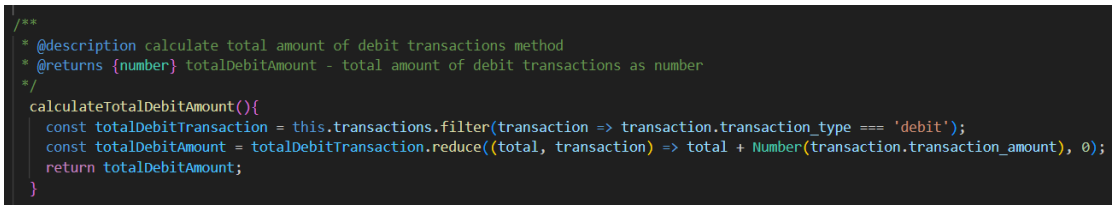
8. Метод getTransactionsByAmountRange(minAmount, maxAmount) - возвращает транзакции с суммой в заданном диапазоне от minAmount до maxAmount



```
/**
 * @description get transactions in given amount range method
 * @param {number} minAmount - min amount
 * @param {number} maxAmount - max amount
 * @returns {Array<string, number>} [transactionString, totalAmount] - transactions as string and total transactions amount as number
 */
 getTransactionsByAmountRange(minAmount, maxAmount){
   const filteredTransactions = this.transactions.filter(transaction => {
     const transactionAmount = Number(transaction.transaction_amount);
     return transactionAmount >= minAmount && transactionAmount <= maxAmount;
   });
   const transactionString = filteredTransactions.map(transaction => transaction.strig()).join('\n');
   const totalAmount = filteredTransactions.reduce((total, transaction) => total + Number(transaction.transaction_amount), 0);
   return [transactionString, totalAmount];
 }
```

Рисунок 13: метод получения транзакций и их суммы в определённом диапазоне

9. Метод calculateTotalDebitAmount() - вычисляет общую сумму дебетовых транзакций



```
/**
 * @description calculate total amount of debit transactions method
 * @returns {number} totalDebitAmount - total amount of debit transactions as number
 */
 calculateTotalDebitAmount(){
   const totalDebitTransaction = this.transactions.filter(transaction => transaction.transaction_type === 'debit');
   const totalDebitAmount = totalDebitTransaction.reduce((total, transaction) => total + Number(transaction.transaction_amount), 0);
   return totalDebitAmount;
 }
```

Рисунок 14: метод рассчёта суммы всех дебетовых транзакций

10. Метод findMostTransactionsMonth() - возвращает месяц, в котором было больше всего транзакций

```
/**
 * @description find month with max transactions method
 * @returns {string} mostTransactionsMonth - month with max transactions as string
 */
 findMostTransactionsMonth(){
   let monthlyRevenue = 0;
   let mostTransactionsMonth = 0;
   const revenueByMonth = this.transactions.reduce((acc, transaction) => {
     const transactionDate = new Date(transaction.transaction_date);
     const transactionMonth = transactionDate.getMonth() + 1;
     if (!acc[transactionMonth]) {
       acc[transactionMonth] = 0;
     }
     acc[transactionMonth] += transaction.transaction_amount;
     return acc;
   }, {});

   for (let month = 1; month <= 12; month++){
     const revenue = revenueByMonth[month] || 0;
     if (revenue > monthlyRevenue) {
       monthlyRevenue = revenue;
       mostTransactionsMonth = month;
     }
   }
   return mostTransactionsMonth;
 }
```

Рисунок 15: метод получения месяца с наибольшим количеством транзакций

11. Метод findMostDebitTransactionMonth() - возвращает месяц, в котором было больше дебетовых транзакций

```
/**
 * @description find month with max debit type transactions method
 * @returns {string} monthWithMostDebit - month with max debit type transactions as string
 */
 findMostDebitTransactionMonth(){
   let maxDebitCount = 0;
   let monthWithMostDebit = 0;
   const transactionCountByMonth = this.transactions.reduce((acc, transaction) => {
     const transactionMonth = new Date(transaction.transaction_date).getMonth() + 1;
     if (transaction.transaction_type === 'debit') {
       acc[transactionMonth] = (acc[transactionMonth] || 0) + 1;
     }
     return acc;
   }, {});
   for (let month = 1; month <= 12; month++){
     const currentCount = transactionCountByMonth[month] || 0;
     if (currentCount > maxDebitCount) {
       maxDebitCount = currentCount;
       monthWithMostDebit = month;
     }
   }
   return monthWithMostDebit;
 }
```

Рисунок 16: метод получения месяца с наибольшим количеством дебетовых транзакций

12. Метод mostTransactionTypes() - возвращает каких транзакций больше всего

```
/**
 * @description find transaction type with max number of transactions occured method
 * @returns {string} debit OR credit OR equal - transaction type with max number of transactions as string
 */
 mostTransactionTypes(){
   let creditTransactions = 0;
   let debitTransactions = 0;
   let equalTransactions = 0;
   for (const transaction of this.transactions) {
     if (transaction.transaction_type === 'credit') {
       creditTransactions++;
     } else if (transaction.transaction_type === 'debit') {
       debitTransactions++;
     } else if (transaction.transaction_type === 'equal') {
       equalTransactions++;
     }
   }
   if (debitTransactions > creditTransactions && debitTransactions > equalTransactions) {
     return 'debit';
   } else if (creditTransactions > debitTransactions && creditTransactions > equalTransactions) {
     return 'credit';
   } else {
     return 'equal';
   }
 }
```

Рисунок 17: метод получения наиболее частого типа транзакций

13. Метод getTransactionsBeforeDate(date) - возвращает транзакции, совершенные до указанной даты

```
/**
 * @description get transactions occured before given date method
 * @param {string} date - date
 * @returns {string} transactionsBeforeDateString - transactions before given date as string
 */
 getTransactionsBeforeDate(date) {
   const dateByMethod = new Date(date);
   const filteredTransactions = this.transactions.filter((transaction) => {
     const transactionDate = new Date(transaction.transaction_date);
     return transactionDate < dateByMethod;
   });
   let transactionsBeforeDateString = filteredTransactions.map(transaction => transaction.string()).join('\n');
   return transactionsBeforeDateString;
 }
```

Рисунок 18: метод получения транзакций до указанной даты

14. Метод findTransactionById(id) - возвращает транзакцию по ее уникальному идентификатору

```
/**
 * @description find transaction by id method
 * @param {string} id - transaction id
 * @returns {string} transactionByIdString - transaction by id as string
 */
 findTransactionById(id) {
   let transactionByIdString = this.transactions.find(transaction => transaction.transaction_id === id).string();
   return transactionByIdString;
 }
```

Рисунок 19: метод получения транзакции по указанному идентификатору

15. Метод mapTransactionDescriptions() - возвращает новый массив, содержащий только описания транзакций

```
/**
 * @description get transaction descriptions method
 * @returns {Array<string>} transactionDescriptionsArray - transaction descriptions as array
 */
  mapTransactionDescriptions() {
    const transactionDescriptionsArray = this.transactions.map(transaction => transaction.transaction_description);
    return transactionDescriptionsArray;
  }
}
```

Рисунок 20: метод получения массива с описаниями транзакций

Ссылка на GIT репозиторий: https://github.com/

Контрольные вопросы

1. Какие примитивные типы данных существуют в JavaScript?
   Примитивные типы данных: number, undefined, boolean, string, symbol, bigint. Также, существует тип данных null, который является специальным примитивом.
2. Какие методы массивов вы использовали для обработки и анализа данных в вашем приложении, и как они помогли в выполнении задачи?
   Метод for…of – способ перебора элементов массива без необходимости прибегать к использованию индексов. Метод map – создаёт новый массив, содержащий результат вызова указанной функции для каждого элемента исходного массива. Метод push – добавление элементов в массив. Метод reduce – применяет функцию к каждому элементу массива и возвращает одно результирующее значение. Метод filter – создаёт новый массив, содержащий все элементы, прошедшие валидацию из переданной функции. Метод find – возвращает значение первого элемента массива, которое соответствует заданным условиям. Метод join – превращает элементы массива в строку, объединяя их.
3. В чем состоит роль конструктора класса?
   Конструктор — это специальный метод, который вызывается при создании нового экземпляра класса. Он инициализирует объект, присваивая значения свойствам. Конструкторы могут принимать параметры, которые используются для инициализации свойств объекта. Это позволяет создавать объекты с различными начальными значениями.
4. Каким образом вы можете создать новый экземпляр класса в JavaScript?
   Классы используются для создания экземпляров. Экземпляр — это объект, содержащий данные и логику класса. Экземпляры создаются с помощью оператора new: instance = new Class().

Список литературы

1. MSU-Courses GitHub – курс JavaScript Государственного Университета Молдовы

2. W3Schools JavaScript Tutorial – базовый курс JavaScript
3. Freecodecamp – курс JavaScript для начинающих
4. «Программирование на JavaScript», Васильев А.Н. – учебное пособие для самостоятельного обучения