Молдавский Государственный Университет

Факультет Математики и Информатики

Департамент Информатики

Отчет по лабораторной работе № 2

по предмету „JavaScript”  
Тема: **Основы работы с массивами, функциями и объектами в JavaScript**

Выполнил: студент группы IA2403

Maria Micikovskaia

Проверил преподаватель:

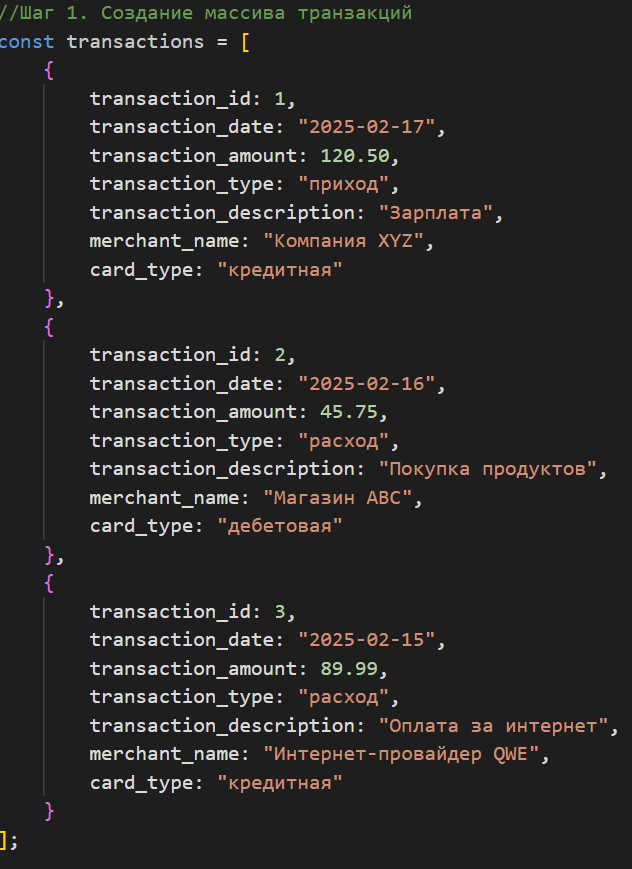
Murinet A., Dr., Conf.univ.

Кишинев, 2024

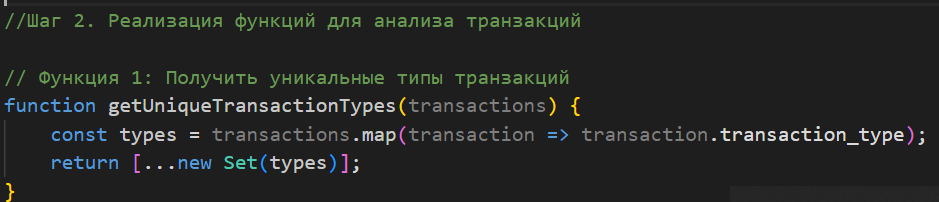
**Цель работы**

Изучить основы работы с массивами и функциями в JavaScript, применяя их для обработки и анализа транзакций.

**Ход работы**



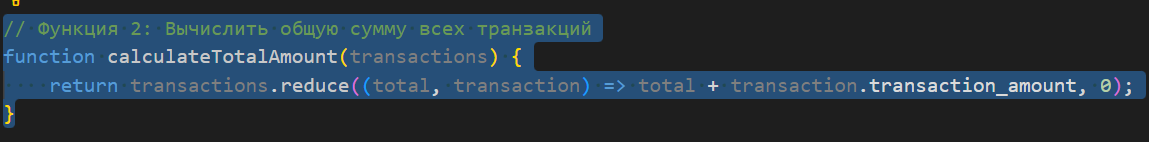
Создаем массив с транзакциями и выводим на экран.



Функция getUniqueTransactionTypes должна вернуть массив уникальных типов транзакций, то есть без повторений.

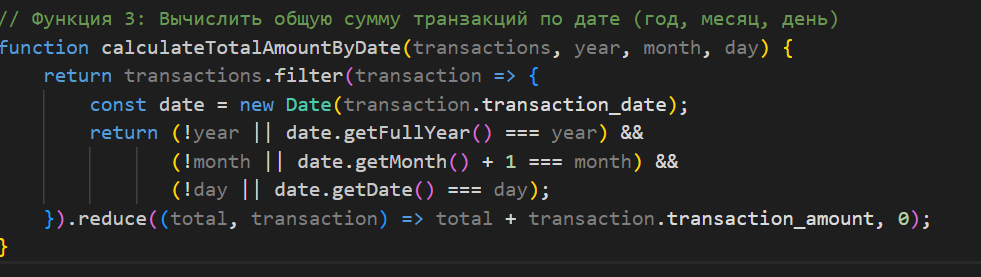
**Что делает каждый шаг:**

1. **transactions.map(transaction => transaction.transaction\_type)**:
   * Метод .map() создает новый массив, содержащий только значения поля transaction\_type из каждого объекта транзакции.
   * Например, если у нас есть массив транзакций, в котором типы транзакций — это ["приход", "расход", "расход"], то после выполнения .map() результат будет: ["приход", "расход", "расход"].
2. **new Set(types)**:
   * Set — это структура данных в JavaScript, которая автоматически удаляет все дубликаты. То есть, если вы создадите новый объект Set из массива, все повторяющиеся элементы будут удалены.
   * Например, если у нас есть массив ["приход", "расход", "расход"], то new Set(types) вернет объект Set с уникальными значениями: Set { "приход", "расход" }.
3. **[...new Set(types)]**:
   * Оператор распространения ... используется для того, чтобы превратить объект Set обратно в массив. В результате мы получаем массив с уникальными типами транзакций.
   * Например, Set { "приход", "расход" } превращается в массив ["приход", "расход"].



**transactions.reduce((total, transaction) => total + transaction.transaction\_amount, 0)**:

* Метод .reduce() применяется к массиву транзакций и используется для аккумулирования (суммирования) значений.



**transactions.filter(transaction => {...}):**

* **Метод .filter() используется для того, чтобы выбрать только те транзакции, которые соответствуют заданным критериям (по году, месяцу и/или дню).**
* **Внутри фильтра мы создаем объект Date из строки transaction.transaction\_date и проверяем его год, месяц и день.**

**const date = new Date(transaction.transaction\_date):**

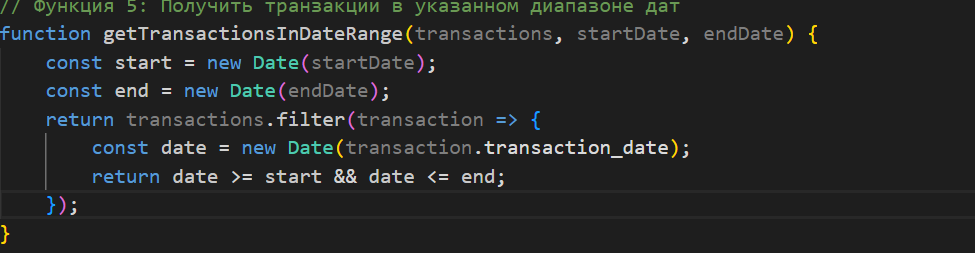
* **Создаем объект Date, который позволяет работать с датой транзакции. transaction.transaction\_date — это строка даты в формате YYYY-MM-DD (например, "2025-02-17").**

**Проверки по году, месяцу и дню:**

* **!year || date.getFullYear() === year: Проверяем, соответствует ли год транзакции заданному году. Если параметр year не передан (равен undefined), то проверка по году пропускается.**
* **!month || date.getMonth() + 1 === month: Проверяем, соответствует ли месяц транзакции заданному месяцу. Метод getMonth() возвращает месяц, начиная с 0 (где 0 — это январь, а 11 — декабрь), поэтому добавляем 1 для правильной проверки.**
* **!day || date.getDate() === day: Проверяем, соответствует ли день транзакции заданному дню.**

**.reduce((total, transaction) => total + transaction.transaction\_amount, 0)**:

* После того как фильтруются транзакции, метод .reduce() суммирует значения transaction.transaction\_amount для каждой оставшейся транзакции.



**const start = new Date(startDate); const end = new Date(endDate);**:

* Мы создаем два объекта Date для начала (startDate) и конца (endDate) диапазона, чтобы легко сравнивать даты транзакций с этим диапазоном.
* Параметры startDate и endDate могут быть строками (например, "2025-02-01") или объектами Date.

**transactions.filter(transaction => {...})**:

* Мы используем метод .filter(), чтобы пройти по всем транзакциям и отфильтровать те, которые соответствуют диапазону дат.
* Внутри фильтра мы создаем объект Date для каждой транзакции (transaction.transaction\_date), чтобы проверить, попадает ли эта транзакция в диапазон между start и end.

**const date = new Date(transaction.transaction\_date);**:

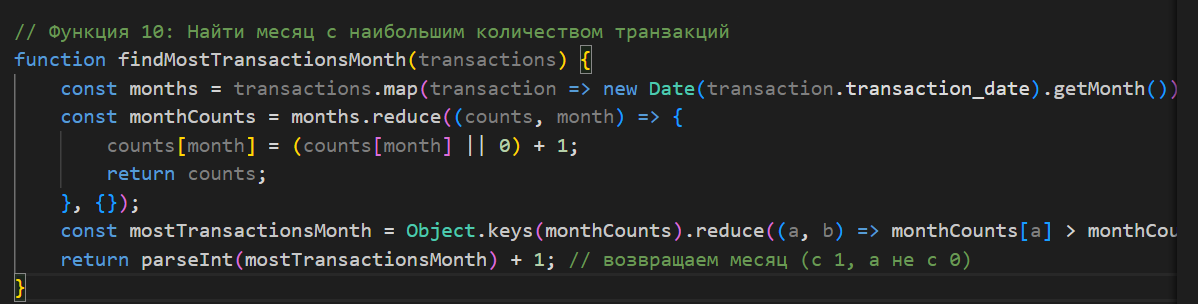
* Для каждой транзакции создаем объект Date, используя строку из поля transaction\_date, которая содержит дату транзакции в формате YYYY-MM-DD.

**date >= start && date <= end**:

* Это условие проверяет, попадает ли дата транзакции в диапазон от start до end.
  + date >= start: дата транзакции должна быть не раньше даты начала диапазона.
  + date <= end: дата транзакции должна быть не позже даты окончания диапазона.

**Возвращаем отфильтрованные транзакции**:

* Функция .filter() возвращает новый массив транзакций, которые соответствуют указанному диапазону дат.



**const months = transactions.map(transaction => new Date(transaction.transaction\_date).getMonth());**:

* Сначала мы используем метод .map() для извлечения месяца из каждой транзакции.
* new Date(transaction.transaction\_date).getMonth() создаст объект Date для каждой транзакции и извлечет месяц с помощью метода getMonth(). Однако getMonth() возвращает число от 0 (январь) до 11 (декабрь), поэтому необходимо будет добавить 1 в конце, чтобы результат был от 1 до 12.

**const monthCounts = months.reduce((counts, month) => {...}, {});**:

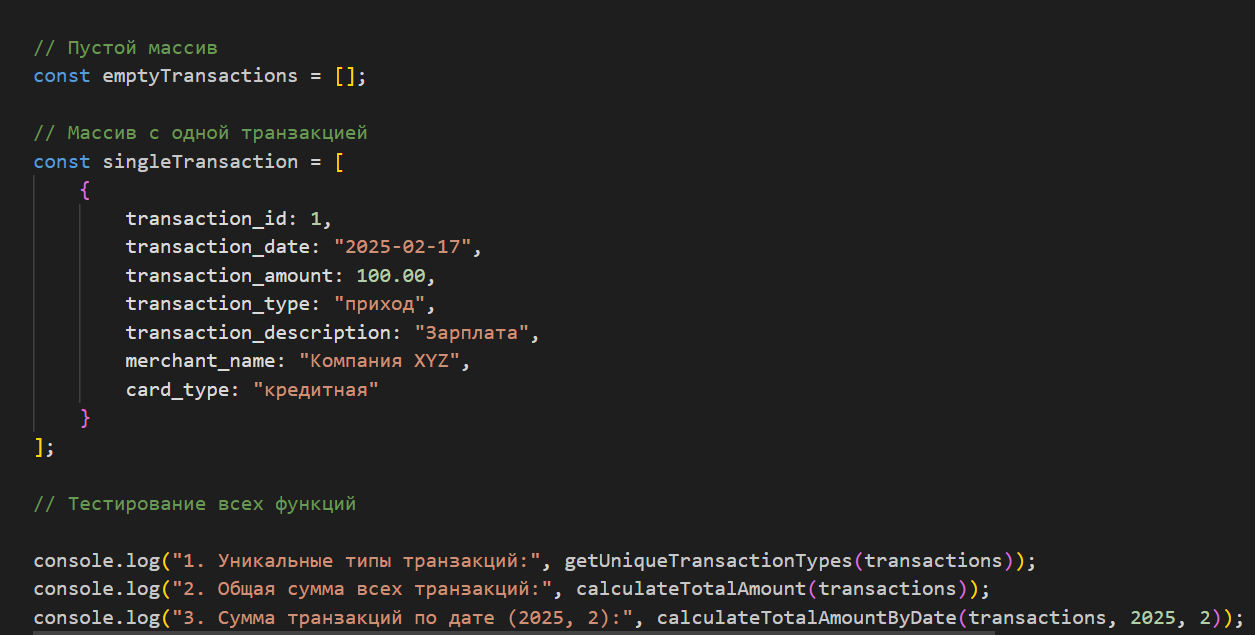
* С помощью метода .reduce() мы создаем объект monthCounts, где ключами будут месяцы (от 0 до 11), а значениями — количество транзакций, произошедших в каждом месяце.
* Внутри .reduce() мы проверяем, сколько раз встречается каждый месяц, и увеличиваем счетчик для соответствующего месяца.

**const mostTransactionsMonth = Object.keys(monthCounts).reduce((a, b) => monthCounts[a] > monthCounts[b] ? a : b);**:

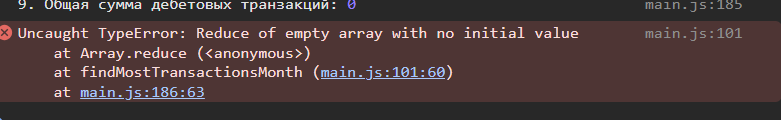
* С помощью Object.keys(monthCounts) получаем все месяцы (в виде строк) из объекта monthCounts.
* Затем, снова применяя .reduce(), мы находим ключ (месяц), с наибольшим количеством транзакций. Мы сравниваем значения (количество транзакций) для каждого месяца, выбирая тот, у которого больше всего транзакций.

**return parseInt(mostTransactionsMonth) + 1;**:

* Поскольку ключи в monthCounts — это строки, нам нужно преобразовать их в числа с помощью parseInt().
* Кроме того, мы добавляем 1 к результату, чтобы получить месяц в привычном формате (1 — январь, 12 — декабрь), так как getMonth() возвращает числа от 0 до 11.



Создаем пустой массив и одиночный, а затем выводим все в консоль.



Ошибка Uncaught TypeError: Reduce of empty array with no initial value возникает, потому что вы пытаетесь использовать метод .reduce() на пустом массиве, а в нем нет начального значения.

Чтобы избежать этой ошибки, можно добавить проверку на пустоту массива перед использованием .reduce()

**Контрольные вопросы:**

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

В JavaScript есть несколько полезных методов для работы с массивами объектов. Вот некоторые из них:

* .map():
  + Применяет функцию ко всем элементам массива и возвращает новый массив с результатами выполнения этой функции.
  + Используется для преобразования каждого объекта в массиве.

.filter():

* Создает новый массив, включающий только те элементы, которые прошли проверку в условии.
* Применяется для отфильтровывания объектов по определенным условиям.

.reduce():

* Применяет функцию-аккумулятор к каждому элементу массива и сводит его к одному значению.
* Используется для вычисления итогового значения на основе массива объектов.
* Пример: Суммировать все возрастающие значения в массиве объектов.

.forEach():

* Применяет функцию ко всем элементам массива, но не возвращает новый массив.
* Используется для выполнения побочных эффектов, например, для вывода в консоль.

.find():

* Возвращает первый элемент массива, который соответствует условию.
* Применяется для поиска первого объекта, удовлетворяющего заданному условию.

.some():

* Проверяет, есть ли в массиве хотя бы один элемент, удовлетворяющий условию.

**2. Как сравнивать даты в строковом формате в JavaScript?**

В JavaScript строки с датами, представленные в формате YYYY-MM-DD (например, "2025-02-17"), могут быть сравнены как обычные строки, поскольку этот формат сортируется лексикографически (по алфавиту) в правильном порядке. Для этого достаточно использовать операторы сравнения (например, <, >, ===).

2. Как сравнивать даты в строковом формате в JavaScript?

**Основное различие**:

* map() используется для преобразования каждого элемента массива в новый массив.
* filter() используется для выбора подмножества элементов, соответствующих определенному условию.
* reduce() используется для свертки массива в одно значение, на основе всех элементов массива.

**Заключение:**

Лабораторная работа позволила укрепить навыки работы с массивами объектов в JavaScript, улучшить понимание обработки данных и научиться применять методы массива для решения практических задач. Эти навыки могут быть полезны при разработке реальных приложений, работающих с большими объемами информации.