Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Ордена Трудового Красного Знамени

федеральное государственное бюджетное

образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра МКиИТ

Проектирование клиент-серверных приложений

Лабораторная работа №7

“Изучение основ JavaScript, создание простейших функций и использование базовых операторов. Работа с элементами DOM с помощью JavaScript”

Выполнил:

студент 3 курса,

группы БФИ2001

Лушин Е. А.

Москва 2023

**Цель работы:** изучение основ JavaScript, создание простейших функций и использование базовых операторов, а также работа с элементами DOM с помощью JavaScript.

**Задание:**

Изучение основ JavaScript, создание простейших функций и использование базовых операторов

* Напишите функцию, которая фильтрует студентов по группе. Функция должна возвращать только тех студентов, что учатся в указанной группе.

Работа с элементами DOM с помощью JavaScript

* Сделайте другую реализацию функции-обработчика, которая уместилась бы в гораздо меньшее количество строк кода. Предложенный выше вариант довольно громоздок, однако он хорошо иллюстрирует некоторые возможности манипулирования DOM-ом с помощью JavaScript. Но весь описанный выше функционал можно реализовать с помощью css-классов и их изменения через функцию-обработчик. Сделайте так, чтобы класс **folded** добавлялся не кнопке, по которой кликнули, а родителю всего поста (это тот самый элемент, у которого в html-разметке установлен класс “**one-post**”). В таком случае свойство display можно изменять более элегантным путем, через css-стили:

*.one-post.folded .article-author{*

*/\* данный стиль применится только для элементов класса*

*.article-author, у которых родитель с классом one-post*

*имеет также класс folded \*/*

*display: none;*

*}*

* По аналогии через css-стили установите исчезновение остальных элементов поста. В функции-обработчике останется только изменять текст кнопки и менять класс для элементов **one-post**.

**Краткая теория**

**Основы языка сценариев JavaScript**

Язык JavaScript — это бесплатный язык сценариев, исполняемых на стороне клиента, который позволяет создавать интерактивные HTML-страницы. «На стороне клиента» (client-side) означает, что JavaScript запускается в Web-браузере и не используется на стороне сервера. Сценарии на стороне клиента позволяют пользователю интерактивно взаимодействовать с Web-страницей после того, как она была обработана сервером и загружена Web-браузером. Например, в GoogleMaps применение языка JavaScript дает пользователям возможность взаимодействовать с картой, перемещать её, приближать и удалять и т. д. Без JavaScript Web-страницу приходилось бы обновлять при каждом взаимодействии с пользователем, если, конечно, не использовать такие плагины, как Adobe Flash или Microsoft Silverlight. Язык JavaScript не требует плагинов.

Так как JavaScript обеспечивает взаимодействие пользователя с Web-страницей после её загрузки, разработчики обычно используют его для решения следующих задач:

* динамическое добавление, редактирование и удаление HTML-элементов и их значений;
* проверка содержимого web-форм перед отправкой на сервер;
* создание на компьютере клиента cookie-файлов для сохранения и получения данных при последующих визитах.

Перед началом изучения языка следует познакомиться с его основными принципами:

* чтобы добавить JavaScript-код в HTML-файл, его необходимо поместить внутрь тегов script, добавить атрибут text/javascript и указать в src путь к файлу;
* все выражения в JavaScript оканчиваются точкой с запятой;
* язык чувствителен к регистру символов;
* имена всех переменных должны начинаться с буквы или символа подчеркивания;
* можно использовать комментарии, чтобы выделить определенные строки в сценарии; комментарии должны начинаться с двойного прямого слеша (//), за которым следует текст комментария;
* комментарии также можно использовать для отключения фрагментов сценария; для отключения нескольких строк можно использовать конструкцию /∗ фрагмент кода ∗/; любой код внутри /∗∗/ не будет запускаться во время выполнения.

Проще всего добавить JavaScript-код на Web-страницу, если загрузить его из внешнего JS-файла с помощью атрибута src в теге script (пример добавления сценария JavaScript к веб-странице):

*<script type=''text/javascript''src=''path/to/javascript-file.js''></script>*

Существуют переменные двух типов: локальные и глобальные. Локальные переменные объявляются с помощью ключевого слова var, а глобальные — без него. При использовании var переменная считается локальной, так как она доступна только в той области, где была объявлена. Например, если объявить локальную переменную внутри функции (этот случай будет рассмотрен в следующих разделах), то к ней нельзя будет получить доступ за пределами функции и она станет локальной переменной данной функции. Если же объявить эту же переменную без var, то она будет доступна по всему сценарию, а не только в этой функции.

Ниже приведен пример создания локальной переменной num, которой присвоено значение 10:

*var num = 10;*

Чтобы сохранить арифметическое выражение в переменной, достаточно присвоить переменной вычисленное значение, как показано в примере ниже. В переменной будет храниться вычисленный результат, а не само выражение.

Пример создания переменной num, как суммы двух чисел:

*var num = (5 + 5);*

В JavaScript операторы требуются для выполнения любого действия — сложения, вычитания, сравнения и т. д. В языке существует четыре типа операторов:

* арифметические:

– сложение: «+»;

– вычитание: «−»;

– умножение: «∗»;

– деление: «/»;

– вычисление остатка от деления: «%»;

– инкремент (увеличение на единицу): «++»;

– декремент (уменьшение на единицу): «−−»;

* присваивания:

– простое присваивание: «=»;

– присвоить переменной результат сложения: «+ =»;

– присвоить переменной результат вычитания: «− =»;

– присвоить переменной результат умножения: «∗ =»;

– присвоить переменной результат деления: «/ =»;

– присвоить переменной результат вычисления остатка от деления: «% =»;

* сравнения:

– равенство: «==»;

– равенство по значению и типу объекта: «===»;

– неравенство: «! =»;

– больше чем: «>»;

– меньше чем: «<»;

– больше или равно: «>=»;

– меньше или равно: «<=»;

* логические:

– И: «&&»;

– ИЛИ: «∣∣»;

– НЕ: «!».

**Массивы**. Массивы похожи на переменные, но отличаются от них тем, что могут хранить несколько значений и выражений под одним именем. Возможность хранения нескольких значений в одной переменой — это главное преимущество массива. В JavaScript для массивов не существует ограничений на количество или тип данных, которые будут в нем храниться, пока эти данные находятся в области видимости массива. Доступ к значению любого элемента массива можно получить в любой момент времени после объявления массива в сценарии. Хотя в JS-массиве можно хранить данные любого типа, включая другие массивы, обычно в массиве хранятся однородные данные и его название также каким-то образом связано с хранящимися данными. Ниже представлены примеры использования массивов для хранения однородных данных.

Пример создания массива:

*var colors = new Array(''orange'', ''blue'', ''red'', ''brown'');*

*// можно создавать массивы с помощью функции Array*

*var shapes = ['' circle'', '' square'', '' triangle'', '' pentagon''];*

*// а можно сразу задавать данные для массива, два способа по*

*// большей части одинаковы*

Хотя доступ к значениям в массиве осуществляется легко, но есть одна тонкость. Массив всегда начинается с 0-го, а не первого элемента, что поначалу может смущать. Нумерация элементов начинается с 0, 1, 2, 3 и т. д. Для доступа к элементу массива необходимо использовать его идентификатор, соответствующий позиции элемента в массиве.

Пример обращения к элементам массива по индексу:

*console.log(''Orange: ''+ colors[0]);*

*console.log(''Blue: ''+ colors[1]);*

*console.log(''Red: ''+ colors[2]);*

*console.log(''Brown: ''+ colors[3])*

Тут для вывода переменных использовалась функция console.log(), которая позволяет узнать значение переменной в панели разработки таких браузеров, как Google Chrome, Mozzila Firefox и других. Чтобы посмотреть на вывод, достаточно включить панель (обычно это клавиша f12) и перейти на вкладку «Console».

Тем же способом можно присвоить значение элементу, находящемуся на определенной позиции в массиве, или обновить значение элемента в массиве. Пример обновления значения массива по индексу:

*var colors = new Array();*

*colors[0] = ''orange'';*

*colors[1] = ''blue'';*

*colors[2] = ''red'';*

*colors[3] = ''brown'';*

*console.log(colors)*

**Ассоциативные массивы, они же объекты**. Список пар ключ:значение. Отличаются от массивов в первую очередь тем, что доступ к определенному элементу производится не за счет позиции в массиве, а за счет имени этого значения. Для создания объектов используются фигурные скобки. Пример создания объектов:

*emptyObject = {};*

*newObject = {‵name′:′John′, ‵age′:30};*

Тип данных для ключа должен быть обязательно строкой, а вот значение может принадлежать к любому классу переменных. Даже функцию можно поместить в качестве значения для одного из элементов объекта. Пример добавления функции

в один из элементов объекта:

*someMan = {‵name′:‵John′, ‘age’:30, ‵sayHello′: function(){*

*console.log(‵Hello!′);*

*}};*

*someMan.sayHello();*

*// выведет: Hello!*

Такие функции, которые не существуют сами по себе, а являются элементом (иначе атрибутом) некоего объекта, называются методы. Так что функция sayHello — это метод объекта someMan.

Чтобы прочитать свойство объекта, можно имя свойства указать через точку либо внутри квадратных скобок. Пример обращения к свойству через точку и через квадратные скобки:

*someMan.name // ‵John′*

*someMan[‵age′] // 30*

Чтобы изменить атрибут объекта, достаточно использовать оператор присваивания для этого атрибута (кстати, если такого атрибута не существует у объекта, то он будет создан, и это главный способ создания новых ключей и их значений).

Пример изменения и создания атрибутов объекта:

*someMan[‵age′] = 31; // теперь Джону стукнул 31*

*someMan[‵occupation′] = ‵doctor′;*

*// и у него появилась профессия — он стал доктором*

**Условия**. Во всех языках программирования основой для создания бизнеслогики являются условные выражения, и JavaScript — не исключение. Условные выражения определяют, какое действие должно быть выполнено в зависимости от условий, установленных в сценарии.

Пример оператора условия:

*var num = 10;*

*if(num == 5)*

*{*

*console.log(''num равно 5'');*

*}*

*else*

*{*

*console.log(''num НЕ равно 5, num равно: ''+ num);*

*}*

**Функции**. Функции обладают целым рядом преимуществ. Во-первых, они служат контейнерами для сценариев, которые будут выполняться только при наступлении события или вызове функции. Также функции не выполняются при первой загрузке Web-страницы, когда Web-браузер загружает и выполняет сценарии, находящиеся на этой странице. Предназначение функции — хранить код для выполнения определенной задачи, чтобы его можно было вызвать в любое время.

Функция оформляется в коде очень просто: она начинается с ключевого слова function, за которым следует пробел и название функции. Название функции может быть любым, но желательно, чтобы оно имело отношение к задаче, которую выполняет функция. Ниже приведен пример функции, изменяющей значение существующей переменной.

Пример создания функции:

*var num = 10;*

*function changeVariableValue() {*

*num = 11;*

*}*

*changeVariableValue();*

*console.log(''num равно: ''+ num);*

Здесь демонстрируются не только структура функции, но и пример её вызова для изменения значения переменной. В представленном фрагменте можно изменить значение переменной, так как она объявлена в области видимости основного сценария, как и сама функция, поэтому функция знает о существовании переменной. Но если объявить переменную внутри функции, то к ней нельзя будет получить доступ за пределами функции.

Функции также могут принимать данные через входные параметры. Функция может иметь один или несколько параметров, и в вызове функции будет передаваться столько параметров, сколько объявлено в сигнатуре функции. Важно не путать понятие «параметр» с «аргументом». Параметр — это часть определения (или сигнатуры) функции, а аргумент — это выражение, используемое при вызове функции. Ниже приведен пример вызова функции, имеющей параметры, и вызова функции с использованием аргументов.

Пример создания функции с параметрами:

*var num = 10;*

*function increase( num)*

*{*

*\_num++;*

*}*

*increase(num);*

*console.log(''num равно: ''+ num);*

*// выведет: num равно: 11*

В функциях также используются выражения return. Эти выражения возвращают значение, полученное после выполнения сценария в функции. Например, можно присвоить переменной значение, возвращенное функцией. Ниже показано, как вернуть значение из функции после выполнения сценария.

Функция, возвращающая сумму переданных ей аргументов:

*function add( num1, num2)*

*{*

*return num1 + num2;*

*}*

*var num = add(10, 30);*

*console.log(''num равно: ''+ num);*

После запуска на странице будет напечатано «num равно 40». Удобство этой функции в том, что можно передать в функцию два параметра, она выполнит их сложение и вернет результат, который и будет присвоен переменной, вместо прямого присваивания.

**Циклы**. Как было продемонстрировано выше, массивы — это удобный способ хранить большое количество многократно используемых данных, но это ещё не всё. Циклы for и while предоставляют средства для итерирования по этим массивам, доступа к их значениям и использования их в сценариях.

Чаще всего в JavaScript используется цикл for. Этот цикл обычно состоит из переменой, которой присваивается числовое значение, затем эта переменная используется с оператором сравнения для проверки условия, а в конце числовое значение переменной увеличивается или уменьшается. В ходе сравнения в цикле for обычно определяется, что числовое значение исходной переменной больше или меньше определенного числового значения. Если это условие выполняется (равно true), то цикл выполняется и значение переменной увеличивается или уменьшается, пока условие не станет равным false. Ниже приведен пример цикла for, работающего, пока числовое значение переменной меньше длины массива.

Пример цикла for:

*var colors = new Array(''orange'', ''blue'', ''red'', ''brown'');*

*for(var i=0; i<colors.length; i++) {*

*console.log(''цвет: ''+ colors[i] + ''<br/>'');*

*}*

Свойство length (длина) у массива содержит числовое значение, равное количеству элементов в массиве. Не забывайте, что индексация массива начинается с 0. Поэтому если в массиве 4 элемента, то значение length равно 4, а доступные индексы — 0, 1, 2, 3.

Другой тип цикла — цикл while. Считается, что этот цикл работает «быстрее» цикла for, но он не очень подходит для итерирования по массивам, и его лучше применять в случаях, когда необходимо выполнять сценарий до тех пор, пока условие истинно. Ниже приведен исходный код цикла while, в котором сценарий выполняется до тех пор, пока значение численной переменной меньше 10.

Пример цикла while:

*var i = 0;*

*while(i < 10)*

*{*

*console.log(i + ''/n''); ///n — символ переноса строки*

*i++;*

*}*

Также в JavaScript довольно большую роль играют глобальные объекты window и document.

Объект window сочетает два в одном: глобальный объект javascript и окно браузера.

Для обращения к функциям и методам window не нужно указывать объект. Обращение к функции decodeURI через объект window и без него:

*window.decodeURI(. . .)*

*// то же что и*

*decodeURI(. . .)*

Вообще, любая переменная в конечном счете (если не найдена локально) ищется в глобальном объекте.

Ко всем глобальным объектам можно обращаться как к свойстам window:

*// можно указать window явно — будет работать:*

*str = new window.String(''test'')*

*a = new window.Array(1,2,3)*

Кроме роли «глобального объекта», window также предоставляет интерфейс для работы с окном браузера.

Функция focus():

*window.focus()*

обычно, если окно не минимизировано, оно делается его текущим и выводится на передний план. Если же окно минимизировано, то его обозначение в списке минимизированных окон начинает мигать.

Функция open():

*var newWin = window.open (strUrl, winName [, winParams])*

Метод open создает новое окно браузера, аналогично команде «Новое окно» в меню браузера. Обычно это не вкладка, а именно новое окно, но в некоторых браузерах можно настроить то или иное поведение явным образом. Если параметр strUrl — пустая строка, то в окно будет загружен пустой ресурс about:blank.

В любом случае, загрузка осуществляется асинхронно. Создается пустое окно, загрузка ресурса в которое начнется уже после завершения исполнения текущего блока кода. Аргументы:

* strUrl — адрес для загрузки в новое окно, любая адресная строка, которую поддерживает браузер.
* winName — имя нового окна.
* winParams — необязательный список настроек, с которыми открывать новое окно.

Функция close():

*window.close()*

закрывает текущее окно. Если закрываемое окно не было открыто при помощи window.open(), то при его закрытии выводится предупреждение. Посетитель может отклонить закрытие.

В свою очередь объект document представляет собой «окно» для работы с DOM вашей HTML-разметки. Если вам будет нужно найти какой-либо элемент в DOM-е, изменить у какого-либо элемента CSS-свойство или удалить целую ветку элементов, вам придется иметь дело с document, даже когда вы этого явно не делаете (например, далее будет описываться библиотека jQuery, которая позволяет не писать много шаблонного кода и сама за программиста работает напрямую с объектом document).

**Выполнение**

Скопируем предыдущий проект. Добавим в проект файл helloworld.js в директории articles/static/js.

Чтобы подключить JavaScript-файл к веб-странице, достаточно вставить следующий тег в html-шаблон: **<script src="{% static 'js/helloworld.js' %}"></script>**

Подключим файл к archive.html. На рисунке 1 показан результат.

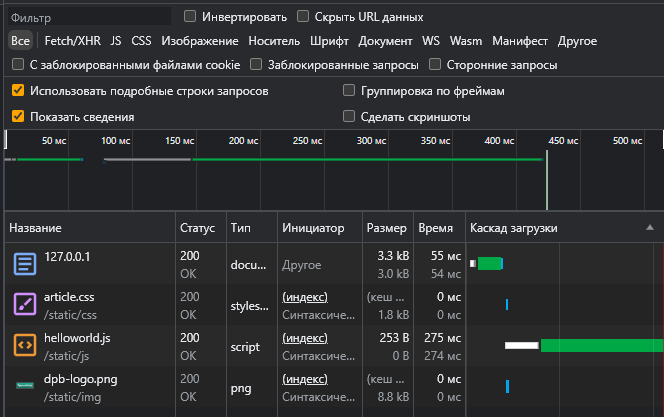
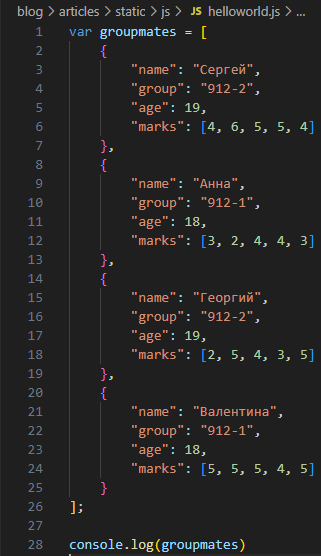
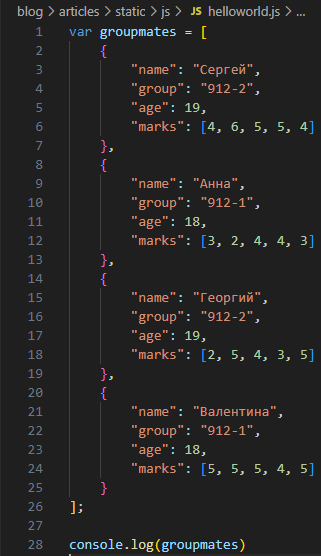


Рисунок 1 – Вкладка “Сеть” консоли разработчика

Добавим в файл код.

Листинг 1. Содержимое файла helloworld.js





Чтобы можно было увидеть значение переменной (иными словами, для вывода переменной на экран), воспользуйтесь методом console.log(), который принимает через запятую аргументы, которые вы желаете увидеть:

*console.log(groupmates);*

На рисунке 2 показан результат.

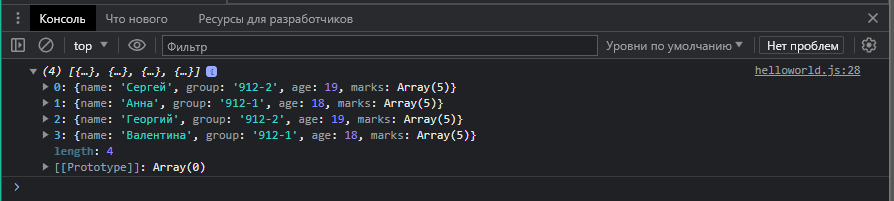
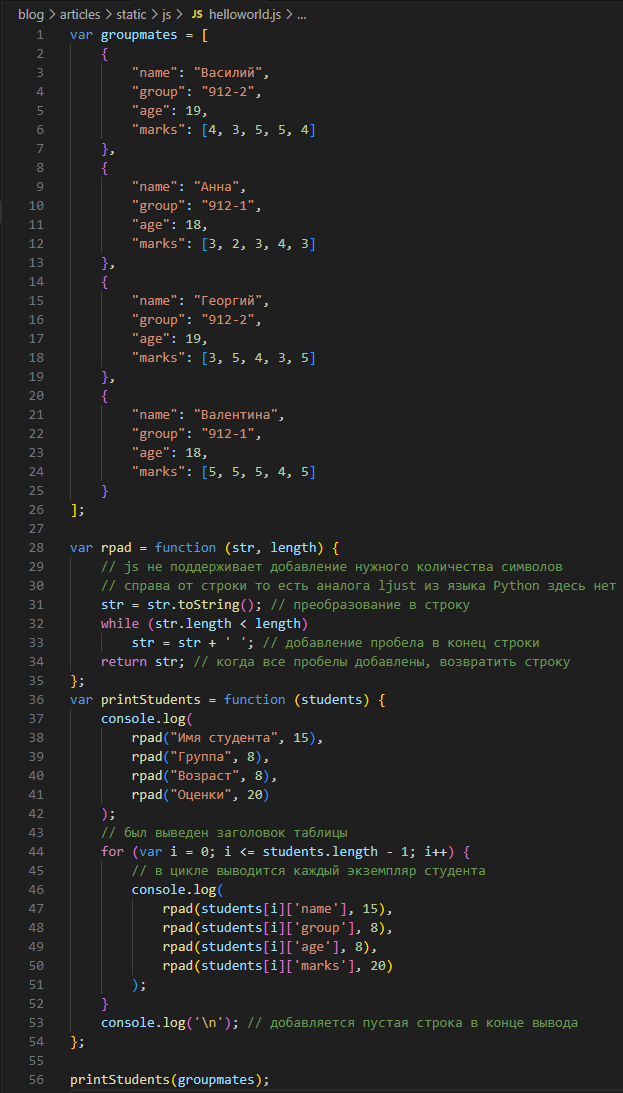
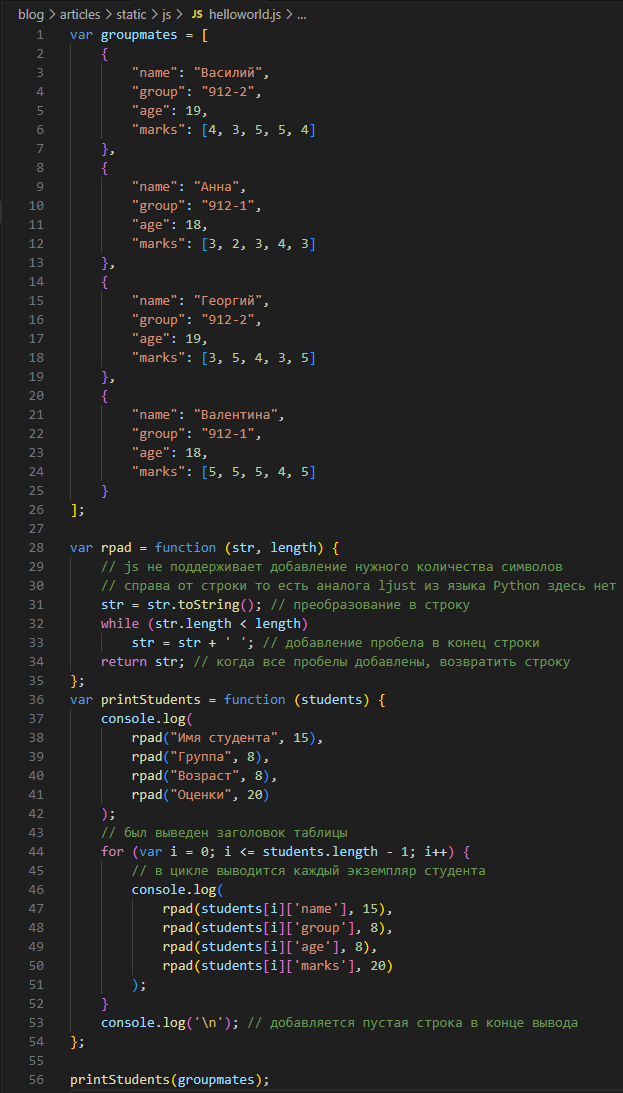


Рисунок 2 – Вывод данных в консоль

Создадим функцию, которая будет выводить в виде таблицы содержимое массива groupmates (вид таблицы должен напоминать тот, что использовался в первой лабораторной работе).

Листинг 2. Содержимое файла helloworld.js





Результат работы представлен на рисунке 3.

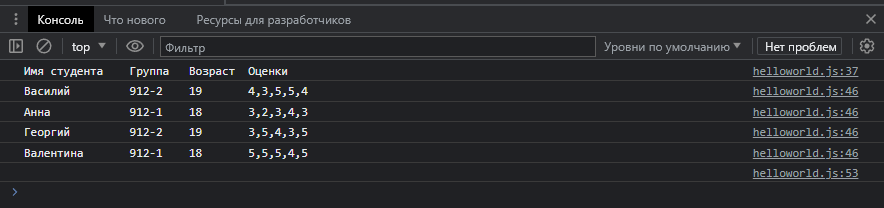


Рисунок 3 – Вывод массива в консоль

Создадим функцию, которая фильтрует студентов.

Листинг 3. Содержимое файла helloworld.js





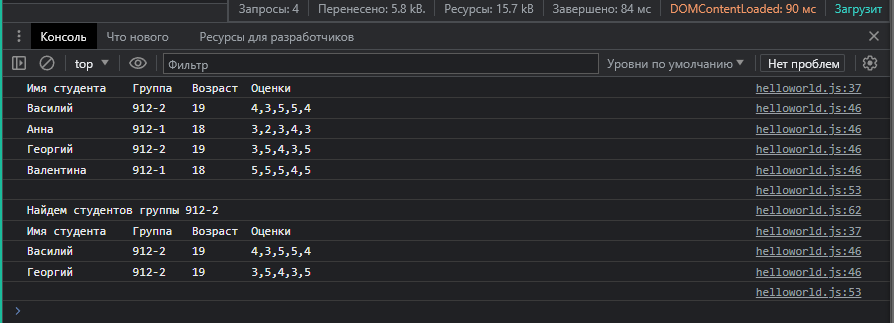


Рисунок 4 – Фильтрация массива groupmates

Добавим кнопки «свернуть» и «развернуть для нашего проекта. Для этого создадим и подключим файл fold-post.js.

Добавим в данный файл событие нажатия кнопки. Поставим точку останова в данное событие. Нажмем на кнопку. Результат представлен на рисунке 5.

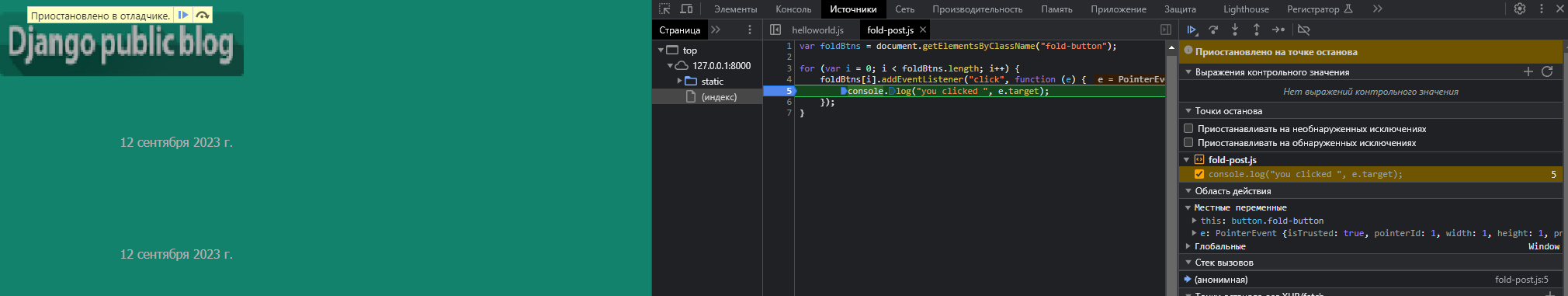
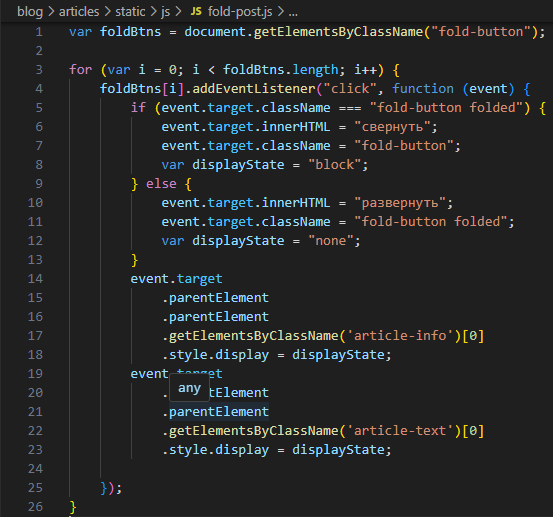


Рисунок 5 – Фильтрация массива groupmates

Добавим код, выполняющий сворачивание/разворачивание информации о статье.

Листинг 4. Содержимое файла fold-post.js



На рисунках 6-7 показан результат на страницах 19 и 20 соответсвенно.



Рисунок 6 – Сворачивание текста



Рисунок 7 – Разворачивание текста

**Вывод:** В данной лабораторной работе мы изучили основы JavaScript, научились создавать простейшие функций и использовать базовые операторы. Также были изучена работа с элементами DOM с помощью JavaScript.