

Браузерный JavaScript. Изменение разметки в документе (конспект)

Дополнительные возможности синтаксиса JavaScript

Как и любой язык программирования, JavaScript менялся со временем. С момента появления в 1995 новые возможности добавлялись каждые несколько лет. В 1997 появился ECMAScript, его целью было направить развитие JavaScript в нужное русло. Выходили новые версии — ES3, ES5, ES6 и так далее, но промежутки между выходами версий длились по 6, а то и по 10 лет. Новая модель стандарта языка предлагает делать маленькие изменения каждый год вместо того, чтобы накопить огромное количество изменений и выпустить их все за раз, как это было с ES6. В ES6 настолько сильно поменялся стандарт языка, что выпускались отдельные справочники именно по ES6 (к примеру, «You Don't Know JS: ES6 & Beyond»).

Версия ES6 вышла в июне 2015 года. Это принесло некую путаницу в название пакета, ведь ES6 и ES2015 — это одно и то же. А поскольку по новой модели стандарта языка обновления выпускались каждый год, ES6 переименовали в ES2015, чтобы отражать год релиза. И все следующие версии теперь называются в соответствии с годом их выпуска.

При разборе примеров в навыке мы будем использовать синтаксис и конструкции, которые появились в ES2015: `const/let`, интерполяция, оператор `spread`, деструктурирующее присваивание, стрелочные функции и другие. Чтобы лучше понимать код и, главное, применять его на практике, рассмотрим некоторые из этих конструкций более детально. Начнём с шаблонных строк и интерполяции.

Шаблонные строки и интерполяция

Шаблонные строки или шаблонные литералы — это строковые литералы, которые допускают использование выражений внутри. С ними вы можете записывать строковые литералы в несколько строк и использовать строковую интерполяцию.

Пример литерала на несколько строк:

```
console.log(`Освещена последняя сосна. Под нею тёмный кряж пушится. Сейчас погаснет и она. День конченный — не повторится.`);
```

Интерполяция строк уже давно поддерживается в других языках программирования, к примеру, в PHP и C#. Вообще интерполяция — это замена переменных-заполнителей или выражений в строке на их значения. Переменные-заполнители и выражения передаются в строку в виде `${...}`

Например:

```
const value = 13; console.log(`Квадрат числа ${value} равен ${value*value}`);  
// Квадрат числа 13 равен 169
```

В выражениях можно использовать условия, например так:

```
<p class="description short"></p>  
const description = document.querySelector('.description'); console.log(`<a  
class="more" href="#">${description.classList.contains('short')} ? 'Читать дальше' :  
'Скрыть описание'</a>`); // "<a class='more' href='#>Читать дальше</a>"
```

Больше информации о строковых литералах можно найти в [спецификации](#).

Деструктурирующее присваивание

Деструктурирующее присваивание (или деструктуризация) в выражениях JavaScript используется, чтобы было проще извлекать данные из массивов или объектов. Разберём на примерах. Начнём с массива.

```
const weekDays = ['Понедельник', 'Вторник', 'Среда', 'Четверг', 'Пятница', 'Суббота', 'Воскресенье'];
```

```
// без деструктуризации
```

```
const monday = weekDays[0];
const tuesday = weekDays[1];
const wednesday = weekDays[2];
const thursday = weekDays[3];
const friday = weekDays[4];
const saturday = weekDays[5];
const sunday = weekDays[6];
```

```
// с деструктуризацией
```

```
const [monday, tuesday, wednesday, thursday, friday, saturday, sunday] = weekDays;
```

С помощью деструктуризации можно быстро поменять местами значения переменных.

```
let first = 1;
let last = 3;
[first, last] = [last, first];
// теперь first содержит значение 3, а last - 1
```

Если массивы деструктуризация разбирает по элементам, то объекты распадутся на свойства.

Допустим, так:

```
const fullName = {name: 'Пётр', surname: 'Петров'};
const {name, surname} = fullName; console.log(`${name} ${surname}`);
// Пётр Петров
```

С помощью деструктурирующего присваивания можно легко разбирать в *JavaScript*-коде *data*-атрибуты. К примеру, у нас есть следующая разметка:

```
<p>Температура воды:</p>
<meter value="0" max="100" low="10" high="60" data-description="Низкая"></meter>
<meter value="30" max="100" low="10" high="60" data-description="Нормальная"></meter>
<meter value="80" max="100" low="10" high="60" data-description="Горячая"></meter>
<meter value="100" max="100" data-description="Кипяток"></meter>
```

Получим значение *data*-атрибута *data-description* для тега `<meter>`:

```
const element = document.querySelector('meter');
const {description} = element.dataset; console.log(`${description}`); // Низкая
```

Подробнее деструктурирующее присваивание описывается в [спецификации](#).

Оператор `spread`

Оператор *spread* (...) или распаковку аргументов удобно использовать при вызове функций с большим числом параметров. Чтобы не перечислять параметры функции через запятую, можно поместить их в массив, а затем распаковать за счёт оператора

Допустим, у нас есть функция для склеивания строк, входящие параметры передаются через запятую:

```
const concat = (str1, str2, str3, str4, str5, str6, str7, str8) => {
  return str1 + str2 + str3 + str4 + str5 + str6 + str7 + str8;
};

const countdown = ['Ты кавай,', 'Я кавай,', 'Мы орём на весь трамвай,', 'Манга,',
'Неки,', 'Сенен-ай,', 'Десу,', 'Ты води давай!!!'];
console.log(concat(...countdown));

// результат:
// Ты кавай,Я кавай,Мы орём на весь трамвай,Манга,Неки,Сенен-ай,Десу,Ты води давай!!!
```

Стрелочные функции

У стрелочных функций в *JavaScript* есть два преимущества перед классическими функциями — это более короткий синтаксис и отсутствие собственного контекста `this`. Плюсы более компактного синтаксиса рассмотрим на примерах.

```
const elements = [
  'Hydrogen',
  'Helium',
  'Lithium',
  'Beryllium'
];

elements.map(function(element) {
  return element.length;
});
// Это выражение вернет массив [8, 6, 7, 9]
```

Функцию в параметрах метода `map` можно записать как стрелочную:

```
elements.map((element) => {
  return element.length;
}); // Получим тот же результат, массив [8, 6, 7, 9]
```

Если единственный оператор в выражении стрелочной функции это `return`, то его можно удалить, а вместе с ним и окружающие фигурные скобки. Так запись функции станет еще компактнее:

```
elements.map(element => element.length); // [8, 6, 7, 9]
```

А теперь попробуем переписать объявление функции:

```
function calendar() {
  return '3 сентября';
};
console.log(calendar());

// эта запись эквивалентна
calendar = () => '3 сентября';
console.log(calendar());
```

И для функциональных выражений применим стрелочную функцию:

```
const multiply = function (a, b) {return a * b};
console.log(multiply(2, 2));
```

```
// эта запись эквивалентна
const multiply = (a, b) => a * b;
console.log(multiply(2, 2));
```

Обо всех особенностях стрелочных функций читайте в [спецификации](#).

Также описание особенностей конструкций языка Java Script можно посмотреть на сайте Mozilla [1], продолжая со страницы, посвященной блокам https://developer.mozilla.org/ru/docs/Learn/JavaScript/Building_blocks

1. Обработка данных

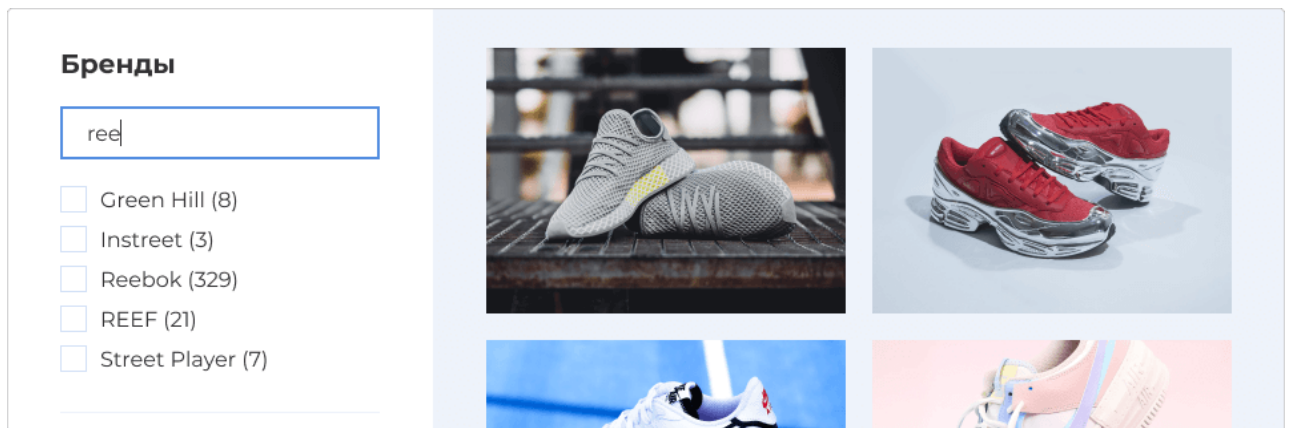
Практически любой JavaScript-код не обходится без обработки коллекций и массивов. Например, вы получили коллекцию DOM-элементов, её нужно в любом случае перебрать и при переборе обработать. Или с сервера пришёл массив банковских терминов для раздела Глоссарий и данные нужно сгруппировать по начальной букве перед выводом и так далее.

Разберем несколько полезных методов для обработки данных.

Группа перебирающих методов массива:

filter — метод создает новый массив, который содержит определённый набор данных из изначально заданного массива.

Метод хорошо подходит, например, для формирования выпадающего списка брендов товаров при вводе символов в поле Бренд.



Фильтрация списка брендов

Чтобы каждый раз не отправлять запрос на сервер, можно фильтровать данные прямо на клиенте.

Например, так:

```
const brands = ['Adidas', 'Green Hill', 'Instreet', 'Reebok', 'REEF', 'Street Player', ...];
```

```
const input = document.querySelector('.input');
const text = input.value;
const list = document.querySelector('.brands');
```

```
// фильтруем список брендов
// на всякий случай всё приводим к одному регистру (верхнему),
// чтобы поиск сделать регистронезависимым
```

```
// формируем HTML строку из тегов li
// а содержимое тега li - название бренда
const itemsString = brands
  .filter((value) => value.toUpperCase().includes(text.toUpperCase()))
  .map((item) => `- ${item}</li>`)
  .join('');

list.insertAdjacentHTML('beforeend', itemsString);

```

О работе с DOM и добавлении элементов с помощью `insertAdjacentHTML` можно узнать в конспекте с описанием [углубленной теории](#).

find — метод для поиска в массиве, вернет первый элемент, который удовлетворяет условию. Условие передается в виде callback функции, например, так:

```
const balls = [
  {
    color: 'red',
  },
  {
    color: 'blue',
  },
  {
    color: 'yellow',
  },
  {
    color: 'aqua'
  }
];

const yellowBall = balls.find((item) => item.color === 'yellow');
```

В результате из массива шаров будет выбран шар, у которого свойство `color` равно строке `'yellow'`, если таких шаров нет, то вернётся значение `undefined`. В нашем случае шар будет найден: `{color: 'yellow'}`.

includes — определяет, содержит ли массив определённый элемент, и в зависимости от этого возвращает `true` или `false`.

```
const pets = ['cat', 'dog', 'bat'];

console.log(pets.includes('cat'));
// вернет true

console.log(pets.includes('raccoon'));
// вернёт false
```

forEach — метод для перебора массива. В современных браузерах с помощью `forEach` можно перебрать элементы коллекций *HTMLCollection* и *NodeList*.

```
const fields = document.querySelectorAll('.field');
// код работает только в современных браузерах
fields.forEach((field) => {
  field.toggleAttribute('disabled');
});
```

О DOM-элементах и коллекциях также рассказано в [углубленной теории](#).

map — метод создает новый массив на основе «старого». А функция в параметре метода описывает как будет видоизменяться каждый элемент исходного массива. Разберём этот метод на классическом примере с подсчетом длины слов в массиве.

```
// исходный массив имён
const rareNames = ['Аполлинария', 'Вилена', 'Иллирика', 'Цецилия',
'Янина'];

// новый массив, в котором элементы - это количество символов в имени
const rareNamesLength = rareNames.map((item) => item.length);
console.log(rareNamesLength);

// [11, 6, 8, 7, 5]
```

join — метод для объединения всех элементов массива в одно строковое значение.

```
const strings = ['Видели ли вы меня в инвизибле?',
'Из инвизибла я б не вылез бы.'];

// объединим строки в одну, при объединении между строками добавим
пробел
console.log(strings.join(' '));
// "Видели ли вы меня в инвизибле? Из инвизибла я б не вылез бы."
```

Рассмотрим ещё один пример с использованием методов **map** и **join** в одной задаче. Добавим на страницу список товаров. Список будет дочерним элементом контейнера `<div class="container">`. В качестве исходных данных приходит массив вида:

```
const cards = [
  {
    title: 'Набор впечатлений «Счастливое мгновение»',
    img: 'img/pic-23.jpg',
    price: 1590,
    description: 'Набор из нескольких впечатлений. Получатель подарка сам
выберет одно понравившееся впечатление и воспользуется им в удобное для
себя время!',
  },
  {...},
  ...
];
```

Сформируем HTML для списка товаров. С помощью метода **map** будем перебирать исходные данные и формировать разметку для элементов списка. А после всё соберём в одну строку, используя метод **join**. Итак, решение:

```
const container =
document.querySelector('.container');

container.innerHTML = '';

// формируем строку с HTML для списка
const catalogString = `<ul class="catalog">
  ${cards.map(({title, img, price, description}) => `
    <li class="author-item">
      <article class="card">
        
        <h3>${title}</h3>
        <p class="price">${price} <a class="cart" href="#">Купить</a></p>
        <p class="description short">${description}</p>
```

```

        </article>
      </li>
    `).join('')}
  </ul>`
);

```

```

container.insertAdjacentHTML('beforeend',
catalogString);

```

Object.entries — метод возвращает массив собственных перечисляемых свойств указанного объекта в формате `[key, value]`. Затем этот массив можно перебрать в цикле `for...of`. Например так:

```

const params = {themeName: 'dark', currentPage: '768900'};

for (const [key, value] of Object.entries(params)) {
  console.log(key, value);
}
// "themeName" "dark"
// "currentPage" "768900"

```

Object.keys — когда нужны только ключи.

```

const params = {themeName: 'dark', currentPage: '768900'};

for (const key of Object.keys(params)) {
  console.log(`ключ: ${key}`);
}

// ключ: themeName , ключ: currentPage

```

Object.values — когда нужны только значения.

```

const params = {themeName: 'dark', currentPage: '768900'};

for (const value of Object.values(params)) {
  console.log(`значение: ${value}`);
}
// значение: dark , значение: 768900

```

2. Использование сторонних библиотек

Иногда дешевле и проще подключить стороннюю библиотеку к сайту, чем изобретать что-то своё. Например, библиотеку для работы с датами [Day.js](#), библиотеку для построения графиков и диаграмм или библиотеку, которая добавляет нужное окончание числительному.

Но при выборе готового решения не забывайте смотреть на некоторые параметры, а именно:

- ➔ когда библиотека обновлялась в последний раз (поддерживают ли разработчики свой продукт, правят ли баги),
- ➔ большой вес библиотеки может замедлять загрузку скриптов,
- ➔ важно наличие документации, в которой описывается, как вызывать методы, хорошо, чтобы были примеры.
- ➔ Сторонние библиотеки еще называют вендорными.
- ➔ Есть несколько способов подключить библиотеку к проекту.

→ Самый простой — подключать библиотеку как обычный js-файл, используя тег `script`.

Разберём этот способ на примере. Подключим к нашему сайту стороннюю библиотеку для правильного склонения числительных `get-declension`. Добавим папку `vendor` и поместим в неё файл `get-declension.min.js`, в котором находится исходный код библиотеки. В `HTML` перед закрывающим тегом `body` добавим строку подключения js-файла. Как правило, сторонние библиотеки подключают перед js-файлами с пользовательским кодом.

```
...  
<script  
src="/base/accessory-03/js/vendor/get-decle  
nsion.min.js"></script>  
<script src="js/script.js"></script>  
</body>
```

После этого в коде можно вызывать функции из подключенной библиотеки:

```
const length = 5;  
const declension = window.getDeclension({count: length, one: 'ступенька', few:  
'ступеньки', many: 'ступенек'});  
  
console.log(`${length} ${declension}`);  
// выведет 5 ступенек
```

При таком способе переменная попадает в глобальную область видимости `window` и нужно быть осторожными, так как можно перетереть другие переменные в `window`. Для изоляции переменных используют импорт модулей (оператор `import`). Но об этом и о других вариантах подключения сторонних библиотек мы расскажем в других навыках.

Методика работы с DOM

«Оживление» страницы за счёт изменения в DOM

Как правило, на динамических сайтах работает два сценария:

- изменение свойств у элементов, которые уже есть на странице,
- добавление новых элементов, которые создаются по заранее известному шаблону.

Обычно оба сценария срабатывают, когда пользователь совершил определенные действия, нажал кнопку, отметил чекбокс и так далее. Иногда действие выполняется при наступлении события (загрузка страницы, окончание загрузки данных и подобные).

Первым делом проанализируем типовые ситуации на сайтах, которые подходят под описанные выше сценарии. А после анализа попробуем составить схему взаимодействия с DOM и выделим основные шаги для реализации нужного функционала.

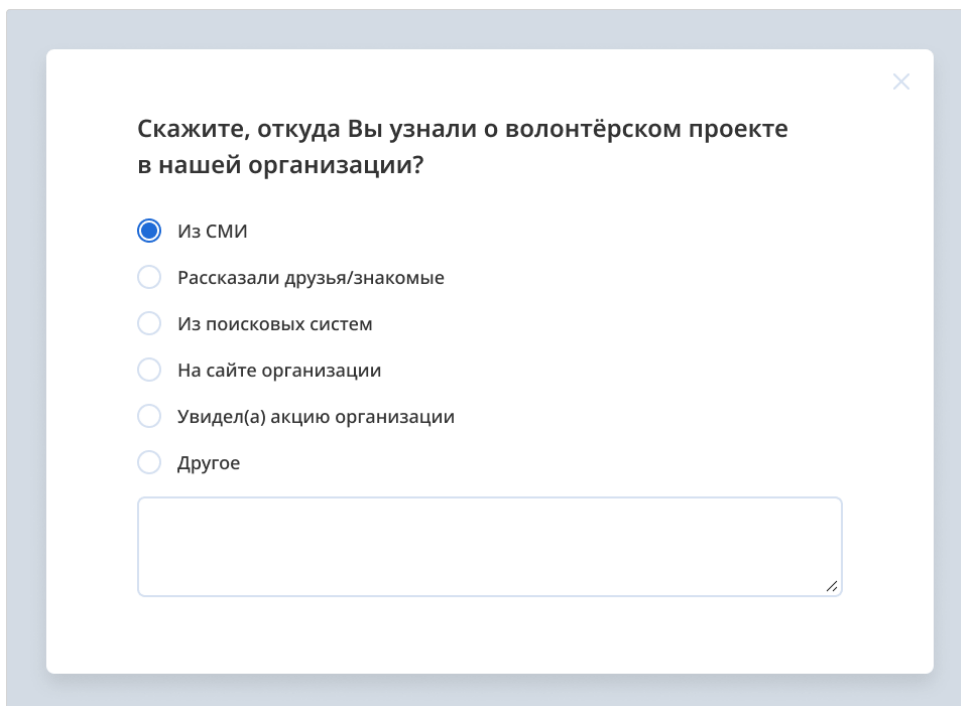
В методике рассмотрим основные схемы «оживления» страницы за счёт JavaScript-кода:

1. при изменении свойств элементов страницы,
2. при добавлении новых элементов в DOM.

1. Изменение свойств элементов страницы

Вот несколько типовых ситуаций.

Первая: пользователь заполняет форму опросника. В пункте «Скажите, откуда Вы узнали о волонтерском проекте в нашей организации?» ему предлагается выбрать только один из вариантов ответа, также есть вариант «Другое», который делает доступным для изменения текстовое поле. По умолчанию текстовое поле заблокировано для ввода.



Форма с вариантом Другое

Получается, что при изменении отметки в группе радиокнопок нужно выяснить, какая из них отмечена. Если это вариант «Другое», снимать атрибут `disabled` с текстового поля, иначе добавлять этот атрибут. Нам понадобится обработчик события `change` у формы. На ней через делегирование будет отлавливать изменение состояния радиокнопок. Но прежде надо найти элементы. Ищем форму и текстовое поле, для которого будем убирать/добавлять атрибут `disabled`.

Разметка для опросника:

```
<form class="feedback" action="#" method="POST">
  <fieldset>
    <legend>Скажите, откуда Вы узнали о волонтерском проекте в нашей
организации?</legend>
    <label>
      <input type="radio" name="source" id="news" value="news" checked>
      Из СМИ
    </label>
    <label>
      <input type="radio" name="source" id="friends" value="friends">
      Рассказали друзья/знакомые
```

```

</label>
<label>
  <input type="radio" name="source" id="search" value="search">
    Из поисковых систем
</label>
<label>
  <input type="radio" name="source" id="site" value="site">
    На сайте организации
</label>
<label>
  <input type="radio" name="source" id="share" value="share">
    Увидел(а) акцию организации
</label>
<label>
  <input type="radio" name="source" id="other" value="other">
    Другое
</label>
<textarea name="other-answer" id="other-answer" disabled>
</textarea>
</fieldset>
</form>

```

Код с решением:

```

// нашли форму
const feedbackForm = document.querySelector('.feedback');
// нашли поле для ввода текста
const textField = document.querySelector('#other-answer');

const formChangeHandler = (evt) => {
  // нас интересует только изменение состояний радиокнопок
  const item = evt.target.closest('input[type="radio"]');

  if (!item) {
    return;
  }
  // если изменяется состояние радиокнопки Другое и она становится отмеченной,
  убираем атрибут disabled
  textField.toggleAttribute('disabled', item.id !== 'other' ||
!item.checked);
}
// подписались на событие изменение формы (через него будем отлавливать
изменение радиокнопок)
feedbackForm.addEventListener('change', formChangeHandler);

```

Другая ситуация

В разделе добавлена внутренняя навигация — список ссылок и закладок. За счёт неё можно сэкономить место на странице. Переключение между закладками выполняет *JavaScript*-код. Активная ссылка может быть выделена с помощью специального CSS-класса, например, *item-active*. При клике на ссылку будем добавлять ей этот класс, при этом убирать его с других ссылок. Скрыть закладки также можно за счёт CSS. Допустим, по умолчанию закладки не показываем, то есть *display: none;*, а для активной устанавливаем *display: block;*. Активную закладку также выделяем специальным классом — *page-active*.

Доставка	Гарантия
Гарантия	Если купленный у нас товар ломается или заискрит, а также в случае пожара, спровоцированного его возгоранием, вы всегда можете быть уверены в нашей гарантии. Мы обменяем сгоревший товар на новый. Дом уж восстановите какнибудь сами.
Кредит	

Внутренняя навигация на странице

Активна закладка *Гарантия*

Доставка	Кредит
Гарантия	Залезть в долговую яму стало проще! Кредитные консультанты придут вам на помощь.
Кредит	Отправить заявку

Переключение на другую вкладку

А теперь переключились на закладку *Кредит*.

Составим разметку для блока. Ссылку и соответствующую ей страницу можно связать через атрибуты `href` и `id`.

Вот, что получилось:

```
<div class="services">
  <ul class="services-navigation">
    <li class="services-navigation-item">
      <a class="services-navigation-link" href="#page-1">Доставка</a>
    </li>
    <li class="services-navigation-item">
      <a class="services-navigation-link" href="#page-2">Гарантия</a>
    </li>
    <li class="services-navigation-item">
      <a class="services-navigation-link item-active"
href="#page-3">Кредит</a>
    </li>
  </ul>
  <div class="services-content">
    <section class="page" id="page-1">
      <h3>Доставка</h3>
      <p>Мы с удовольствием доставим ваш товар прямо к вашему подъезду совершенно бесплатно! Ведь мы неплохо заработаем, поднимая его на ваш этаж!</p>
    </section>
    <section class="page" id="page-2">
      <h3>Гарантия</h3>
      <p>Если купленный у нас товар ломается или заискрит, а также в случае пожара, спровоцированного его возгоранием, вы всегда можете быть уверены в нашей гарантии. Мы обменяем сгоревший товар на новый. Дом уж восстановите какнибудь сами.</p>
    </section>
    <section class="page page-active" id="page-3">
      <h3>Кредит</h3>
      <p>Залезть в долговую яму стало проще! Кредитные консультанты придут
```

```

вам на помощь.</p>
    <a href="#">Отправить заявку</a>
  </section>
</div>
</div>

```

Опишем стили:

```

...
.services-navigation-link {
  padding: 10px 15px;
  color: inherit;
}

/* активная ссылка */
.services-navigation-link.item-active {
  text-decoration: none;
}

/* по умолчанию страница скрыта */
.services-content section {
  display: none;
}

/* активная страница */
.services-content .page-active {
  display: block;
}

```

В *JavaScript*-коде найдем элемент с навигацией. Также найдем элементы с ссылками и элементы со страницами. Для элемента с навигацией добавим обработчик события **click** (снова используем делегирование, чтобы не создавать много обработчиков событий). А дальше всё просто: при клике на ссылку добавляем ей класс **item-active** и удаляем его у других ссылок. С помощью метода **getAttribute** узнаем идентификатор страницы, которая станет активной. По этому идентификатору добавляем класс **page-active** для будущей активной страницы и удаляем его у других страниц.

```

// Поиск элементов
// Навигация
const nav = document.querySelector('.services-navigation');
// отдельные ссылки в навигации
const navItems = document.querySelectorAll('.services-navigation-link');
// элементы со страницами
const pages = document.querySelectorAll('.page');

const setActivePage = (anchor) => {
  // делаем активным пункт меню (переключаем стили), ориентир - значение
  атрибута href
  navItems.forEach((navItem) => {
    navItem.classList.toggle('item-active', anchor ===
    navItem.getAttribute('href'));
  });

  // показываем страницу (переключаем стили), на которую ссылается пункт меню
  pages.forEach((page) => {
    page.classList.toggle('page-active', anchor === ('#' + page.id));
  });
}

```

```

const navClickHandler = (evt) => {
  // используем делегирование и обрабатываем Click только на ссылках в навигации
  const link = evt.target.closest('a');

  if (!link) {
    return;
  }

  // получаем значение якоря, по нему будем открывать соответствующую страницу
  const anchor = link.getAttribute('href');

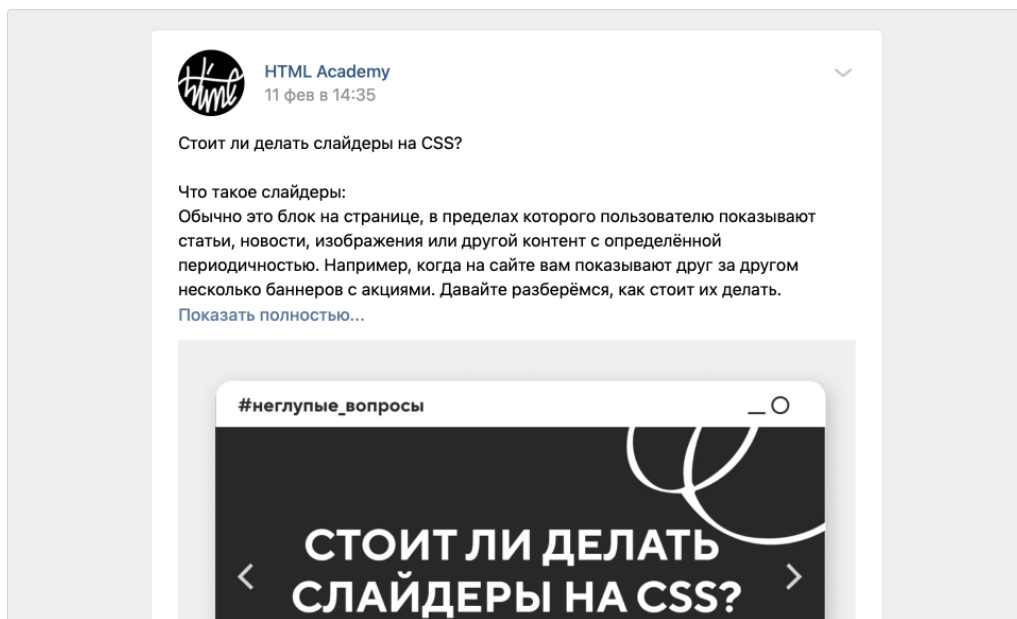
  // переключение страницы по значению якоря
  setActivePage(anchor);
}

const init = () => {
  // подписываемся на событие Click в навигации
  nav.addEventListener('click', navClickHandler);
}

// начальная инициализация элементов страницы
init();

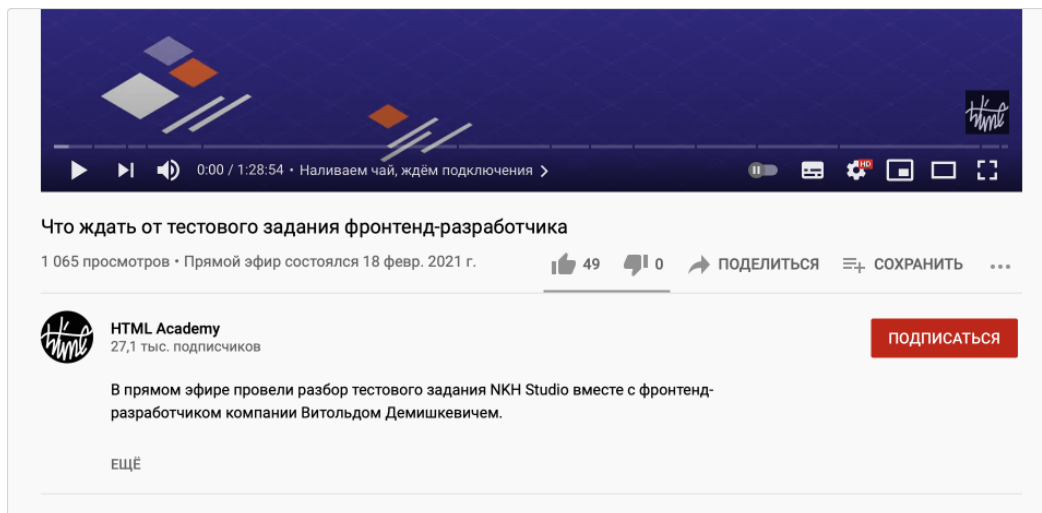
```

*Еще одна типовая ситуация: карточки с объёмным текстовым содержимым и ссылкой **Показать полностью....***

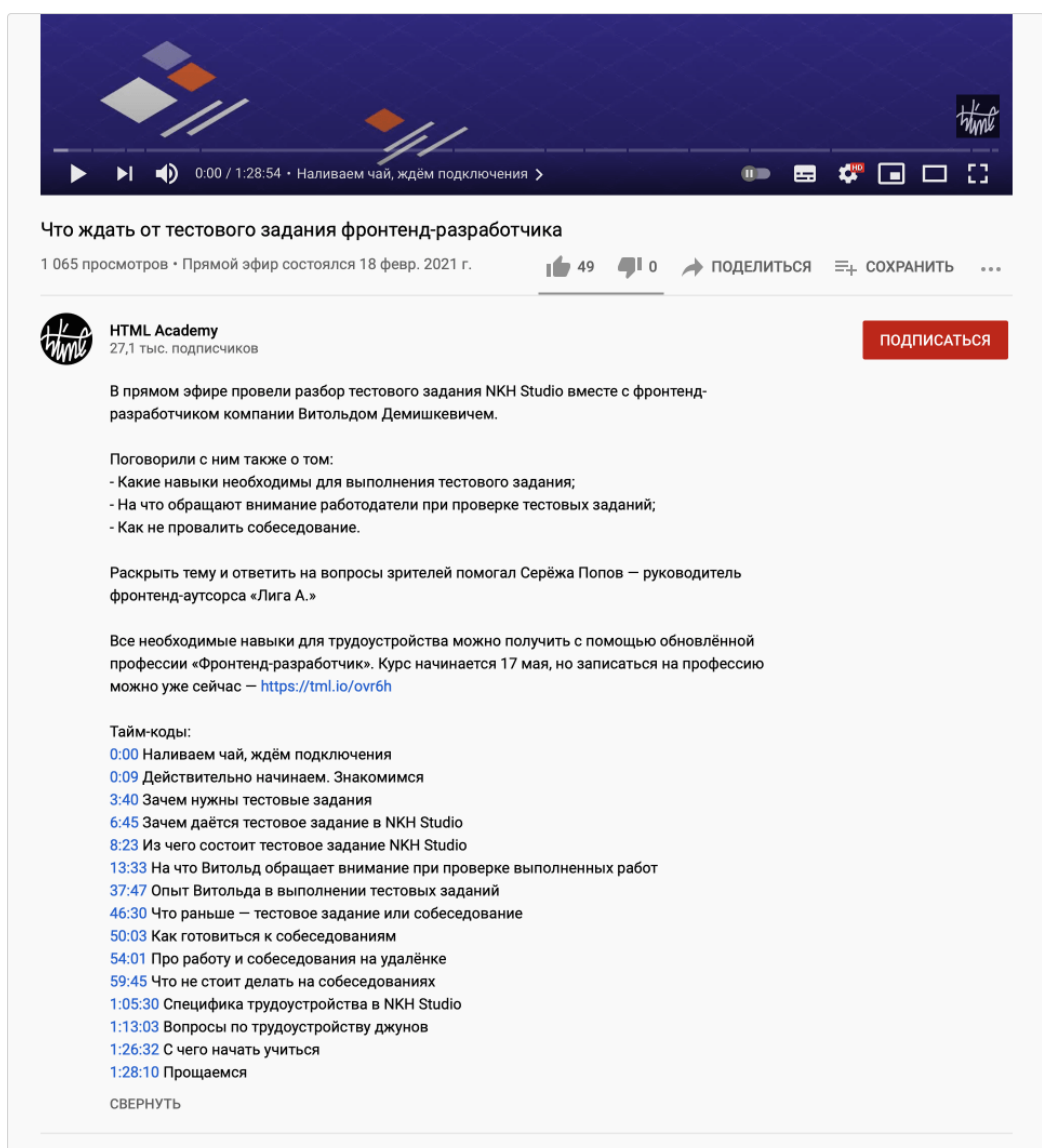


Элемент Показать полностью... на странице

Или так: подробную информацию можно показать, а затем снова свернуть.



Элемент Ещё



Элемент Свернуть

Реализуем второй вариант, то есть будем скрывать и показывать полное описание по клику на ссылку **Ещё/Свернуть**. Скрыть описание можно за счёт стилей — установить максимальную высоту блока, а всё, что не поместилось, обрезать при помощи **overflow: hidden;**. Опишем CSS-класс `description-short` с соответствующими свойствами. При клике на ссылку будет переключаться

ЭТОТ КЛАСС И ДОПОЛНИТЕЛЬНО МЕНЯТЬСЯ ТЕКСТ ССЫЛКИ, В ЗАВИСИМОСТИ ОТ ТОГО, КАКОЕ СЕЙЧАС ОПИСАНИЕ — ПОЛНОЕ ИЛИ СОКРАЩЕННОЕ.

Итак, разметка:

```
<div class="content">
  <div class="description description-short">
    <p>В прямом эфире провели разбор тестового задания NKN Studio вместе с
    фронтенд-разработчиком компании Витольдом Демишкевичем.</p>
  </div>
  <a class="more" href="#">Ещё</a>
</div>
```

Стили:

```
.content {
  padding: 20px 0;
  width: 570px;
}

/* класс для свёрнутого описания */
.description-short {
  overflow: hidden;
  max-height: 60px;
}
```

Код:

```
const content = document.querySelector('.content');

const moreLinkClickHandler = (evt) => {
  // отлавливаем событие только на ссылке с классом more
  const element = evt.target.closest('a.more');

  if (!element) {
    return;
  }
  // отменяем для ссылки действие по умолчанию
  evt.preventDefault();

  const descriptionElement = document.querySelector('.description');

  if (descriptionElement) {
    // переключаем класс
    descriptionElement.classList.toggle('description-short');
    // меняем текст ссылки
    element.textContent =
      descriptionElement.classList.contains('description-short') ? 'Ещё' :
      'Свернуть';
  }
};
content.addEventListener('click', moreLinkClickHandler);
```

Если не вдаваться в подробности, то схема работы во всех описанных случаях одинаковая:

1. сначала — поиск элемента,
2. затем изменение свойств у элемента (это может быть добавление/удаление класса, добавление/удаление атрибута, изменение содержимого элемента и так далее)

2. Добавление новых элементов в DOM

Добавиться может группа однотипных элементов, например, новости в ВКонтакте.



Архитектура

28 янв в 15:00

Курсовой проект "Паркинг на 300 мест" АСИ УГНТУ, 3 курс.



Burton

29 янв в 19:00

Сноупарк в черте города Новосибирск — уникальное место, которое когда-то дало нам несколько очень хороших райдеров.

Новости в ВКонтакте

Разберем эту ситуацию подробнее. Новые посты появляются ниже уже существующих (бесконечная загрузка и вставка после). Как это работает в упрощённом варианте.

У нас есть контейнер с записями `<div class="posts">`. В нём содержатся блоки с отдельными новостями `<section class="post">`. Опишем разметку блоков:

```
<div class="posts">
  <section class="post">
    <header class="post-header">
      <h2>Архитектура</h2>
      
      <time datetime="2021-01-28 15:00">28 янв в 15:00</time>
    </header>
    <div class="post-content">
      <p>Курсовой проект "Паркинг на 300 мест" АСИ УГНТУ, 3 курс</p>
    </div>
  </section>
```

```

<section class="post">
  <header class="post-header">
    <h2>Burton</h2>
    
    <time datetime="2021-01-29 19:00">29 янв в 19:00</time>
  </header>
  <div class="post-content">
    <p>Сноупарк в черте города Новосибирск - уникальное место, которое когда-то
    дало нам несколько очень хороших райдеров.</p>
  </div>
</section>
</div>

```

Сначала с помощью шаблонных строки и интерполяции сформируем HTML для элементов. А затем для добавления их в DOM используем метод `insertAdjacentHTML`.

```

const posts = [
  {
    title: '"Мастера"',
    date: '2021-01-29 20:00',
    img: 'img/pic-21.jpg',
    text: 'Castella Shop - это единственная в Санкт-Петербурге пекарня, которая
    специализируется на приготовлении кастеллы - нежнейшего хлопкового японского
    бисквита',
  },
  {
    title: 'Секрет Фирмы',
    date: '2021-01-29 21:00',
    img: 'img/pic-25.jpg',
    text: 'Одного хештега в Twitter главы Tesla и SpaceX оказалось достаточно,
    чтобы инвесторы кинулись купить криптовалюту',
  }
];

```

```

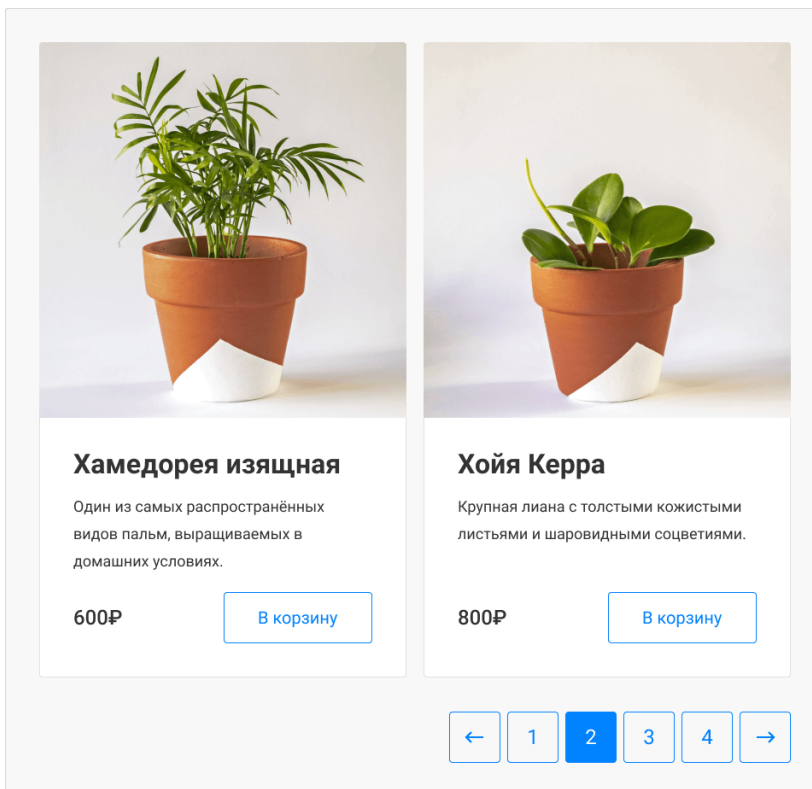
// находим контейнер с постами
const postsContainer = document.querySelector('.posts');

// формируем элементы с новыми постами
// используем шаблонные строки и интерполяцию
// склеиваем всё в одну строку
const postsString = posts.map(({title, date, img, text}) => `
  <section class="post">
    <header class="post-header">
      <h2>${title}</h2>
      
      <time datetime="${date}">${date}</time>
    </header>
    <div class="post-content">
      <p>${text}</p>
    </div>
  </section>
`).join('');

// добавляем посты в контейнер
postsContainer.insertAdjacentHTML('beforeend', postsString);

```

Всем знакомая пагинация (или пейджинация) — это тоже создание новых элементов, которые заменяют существующие.



Пагинация на странице

Рассмотрим этот кейс на примере интернет-магазина по продаже цветочных растений. Разметка отдельной страницы с карточками может выглядеть так:

```
<div class="cards">
  <section class="card">
    <h2>Хамедорея изящная</h2>
    
    <p>Один из самых распространённых видов пальм, выращиваемых в домашних условиях.</p>
    <p>600 руб</p>
    <a href="#">В корзину</a>
  </section>
  <section class="card">
    <h2>Хойя Керра</h2>
    
    <p>Крупная лиана с толстыми кожистыми листьями и шаровидными соцветиями.</p>
    <p>800 руб</p>
    <a href="#">В корзину</a>
  </section>
</div>
```

При переключении со страницы на страницу содержимое элемента `<div class="cards">` очищается и взамен добавляются данные для выбранной страницы.

Также как и в предыдущем случае, для создания элементов используем шаблонные строки и интерполяцию и метод `insertAdjacentHTML` для добавления в DOM. Но предварительно будем очищать родительский элемент от содержимого.

Напишем код, который реализует переключение между страницами.

```

const cards = [
  {
    title: 'Хамедорея изящная',
    img: 'img/pic-34.jpg',
    price: '600',
    description: 'Один из самых распространённых видов пальм, выращиваемых в домашних условиях.',
  },
  {
    title: 'Хойя Керра',
    img: 'img/pic-37.jpg',
    price: '800',
    description: 'Крупная лиана с толстыми кожистыми листьями и шаровидными соцветиями.',
  },
  ...
];

// находим родительский элемент с карточками
const cardsContainer = document.querySelector('.cards');

// очищаем содержимое <div class="cards">
cardsContainer.innerHTML = '';

// формируем HTML с новыми карточками
// используем шаблонные строки и интерполяцию
const cardsString = cards.map(({title, img, price, description}) => `
  <section class="card">
    <h2>${title}</h2>
    
    <p>${description}</p>
    <p>${price}</p>
    <a href="#">В корзину</a>
  </section>
`).join('');

// заполняем элемент <div class="cards"> новыми данными
cardsContainer.insertAdjacentHTML('beforeend', cardsString);

```

Добавляться могут одиночные или разнотипные элементы. К примеру, в форме регистрации пользователь может добавить верификацию по номеру телефона.

Ваш E-mail:

Пароль:

Верификация по номеру телефона ☐

Зарегистрироваться

Ваш E-mail:

Пароль:

Верификация по номеру телефона ☒

Телефон:

Зарегистрироваться

Дополнительная верификация по номеру телефона

Поле с телефоном добавляем по нажатию на переключатель «Верификация по телефону».

```
<form class="registration" action="#" method="POST">
  <label>
    Ваш E-mail:
    <input type="email" name="email" placeholder="ivanov@mail.ru">
  </label>
  <label>
    Пароль:
    <input type="password" name="password" placeholder="password">
  </label>
  <label>
    Валидация по номеру телефона
    <input type="checkbox" class="more">
  </label>
  <div class="verification-container">
  </div>
  <button type="submit">Зарегистрироваться</button>
</form>

// находим родительский элемент для новых полей
const verificationContainer = document.querySelector('.verification-container');

// переключатель дополнительных полей
const switchButton = document.querySelector('.more');

switchButton.addEventListener('click', (evt) => {
  // предварительно почистим родительский элемент
  verificationContainer.innerHTML = '';

  if (switchButton.checked) {
    // используем шаблонные строки
    const fieldsString = `
    <label>
      Телефон:
      <input type="phone" name="phone" placeholder="+7(999)999-99-99">
    </label>`;

    // добавляем поля в родительский элемент
    verificationContainer.insertAdjacentHTML('beforeend', fieldsString);
  }
});
```

После анализа описанных кейсов мы можем выделить следующие шаги:

- ➔ Ищем нужный нам *DOM*-элемент на странице. Этот элемент будет родительским, целевым элементом или точкой отсчёта.
- ➔ Чистим прежнее содержимое найденного целевого элемента `innerHTML = ''`. Этот пункт выполняем, если требуется заменить прежнее содержимое элемента.
- ➔ Создаём новый (ые) элемент (ы). Используем методы: `document.createElement`, тег `<template>`, `createContextualFragment` или *HTML*-строки.
- ➔ Устанавливаем значение атрибутов, заполняем содержимое, если требуется. Используем методы: `setAttribute`, `textContent`, `innerHTML` и другие.
- ➔ Добавляем созданный (е) элемент (ы) в *DOM*. Используем элемент, который нашли в первом пункте как родительский, целевой элемент или точку

отсчета, и для этого элемента вызываем методы `appendChild`, `append`, `insertAdjacentElement`, `insertBefore` или `insertAdjacentHTML`.

Выбор методов зависит от того, какая браузерная поддержка нужна, добавляется один или несколько элементов, а также есть ли пользовательский ввод.

3. Рефакторинг и оптимизация кода

Прежде чем приступить к рефакторингу и оптимизации нашего кода, определимся, что же такое рефакторинг.

Рефакторинг — это процесс такого изменения кода, при котором не меняется внешнее поведение системы, а улучшается лишь его внутренняя структура. По сути, вы улучшаете дизайн кода, после того как он написан. Ведь хорошо структурированный код легко читается и быстро дорабатывается.

Часто в процессе меняются требования к задаче, находятся баги, которые нужно быстро исправить, или возникают срочные доработки. В итоге даже хорошо структурированный код становится беспорядочным и непонятным.

Еще одна задача рефакторинга — сделать код универсальным, так, чтобы при появлении нового функционала не приходилось дорабатывать код. Ведь когда мы реализуем новый функционал, мы стараемся сделать код понятным, но при этом он не всегда универсален. По мере роста числа функций начинаем замечать, что многие из них, хоть и с некоторыми отличиями, повторяют предыдущий код.

Чтобы избежать дублирования кода также нужен рефакторинг.

Попробуем на примере пройти все стадии написания и улучшения кода, от простого и понятного, но не универсального и громоздкого, к, возможно, менее понятному, но компактному и универсальному, готовому к добавлению нового функционала.

Задача: нужно реализовать простой редактор, который применяет выбранное форматирование к тексту.

Подобный редактор есть в Instagram Stories при добавлении текстового контента.

На панели инструментов редактора расположены кнопки **B**, **I**, **U**. Эти кнопки преобразуют текст в редакторе в текст с жирным начертанием, курсивный текст или текст с подчеркиванием. Повторный клик по кнопке снимает соответствующее форматирование. Если первый раз нажать на кнопку **B**, текст будет выделен жирным начертанием, если второй раз нажать на кнопку **B**, жирное начертание снимается.

B **/** **U**

Около тополя снег был примят, местами изрыт, а между деревом и землёй что-то виднелось. Мы остановились в нерешительности.

Редактор текстового контента

B ***/*** **U**

Около тополя снег был примят, местами изрыт, а между деревом и землёй что-то виднелось. Мы остановились в нерешительности.

Выбрано курсивное начертание для текста

Опишем разметку для редактора:

```
<section class="editor">
  <h2 class="visually-hidden">Редактор</h2>
  <ul class="editor-toolbar toolbar">
    <li>
      <button type="button" class="toolbar-button toolbar-button-bold">B</button>
    </li>
    <li>
      <button type="button" class="toolbar-button toolbar-button-italic">I</button>
    </li>
    <li>
      <button type="button" class="toolbar-button toolbar-button-underline">U</button>
    </li>
  </ul>
  <p class="text">Около тополя снег был примят, местами изрыт, а между деревом
и землей что-то виднелось. Мы остановились в нерешительности.</p>
</section>
```

Добавим необходимые стили:

```
...
.text {
  width: 500px;
  padding: 20px;
  font-size: 24px;
  line-height: 1.5;
  border: 1px solid #421eac;
}

.bold {
  font-weight: 600;
}
```



```
.italic {
  font-style: italic;
}

.underline {
  text-decoration: underline;
}
```

Первый вариант реализации кода. Для каждой кнопки у нас будет свой обработчик события.

```
// находим кнопки для форматирования
const boldButton = document.querySelector('.toolbar-button-bold');
const italicButton = document.querySelector('.toolbar-button-italic');
const underlineButton = document.querySelector('.toolbar-button-underline');

// находим текст
const text = document.querySelector('.text');

// определяем обработчики событий для кнопок
boldButton.addEventListener('click', () => {
  // переключаем класс, который соответствует кнопке
  text.classList.toggle('bold');
});

italicButton.addEventListener('click', () => {
  // переключаем класс, который соответствует кнопке
  text.classList.toggle('italic');
});

underlineButton.addEventListener('click', () => {
  // переключаем класс, который соответствует кнопке
  text.classList.toggle('underline');
});
```

В коде всё просто и понятно, но не универсально. Проблема этого кода в том, что он довольно **императивен**. То есть мы явно указываем, что такая-то кнопка при таком-то значении меняется на такое-то. Это, вроде как, неплохо, но код становится негибким. К примеру, одна управляющая кнопка убирается или еще одна добавится в разметку — и *JavaScript*-код придётся переписывать.

Попробуем прийти к состоянию кода, который будет:

1. Устойчив к изменению количества управляющих кнопок.
2. Не будет хранить в себе знание о том, что это за кнопки конкретно. К примеру, функцию, которую выполняет кнопка, можно вынести в **data**-атрибуты (**data-setting**) и добавить в разметку.

В этом случае мы можем использовать делегирование и создать один обработчик для всех кнопок. Предварительно добавим контейнеру с кнопками специальный класс **js-buttons-container**, чтобы отделить классы для стилей и классы для *JavaScript*. Используем метод **closest**, чтобы быть уверенными, что событие произошло именно на кнопке.

Поправим разметку и перепишем наш код.

```
<section class="editor">
  <h2 class="visually-hidden">Редактор</h2>
  <ul class="editor-toolbar toolbar js-buttons-container">
```

```

    <li>
      <button type="button" data-setting="bold" class="toolbar-button
toolbar-button-bold">B</button>
    </li>
    <li>
      <button type="button" data-setting="italic" class="toolbar-button
toolbar-button-italic">I</button>
    </li>
    <li>
      <button type="button" data-setting="underline" class="toolbar-button
toolbar-button-underline">U</button>
    </li>
  </ul>
  <p class="text">Около тополя снег был примят, местами изрыт, а между деревом и
землей что-то виднелось. Мы остановились в нерешительности.</p>
</section>

// находим панель с кнопками
const toolbar = document.querySelector('.js-buttons-container');

// находим текст
const text = document.querySelector('.text');

// определяем обработчики событий для кнопок
toolbar.addEventListener('click', (evt) => {
  const button = evt.target.closest('button');

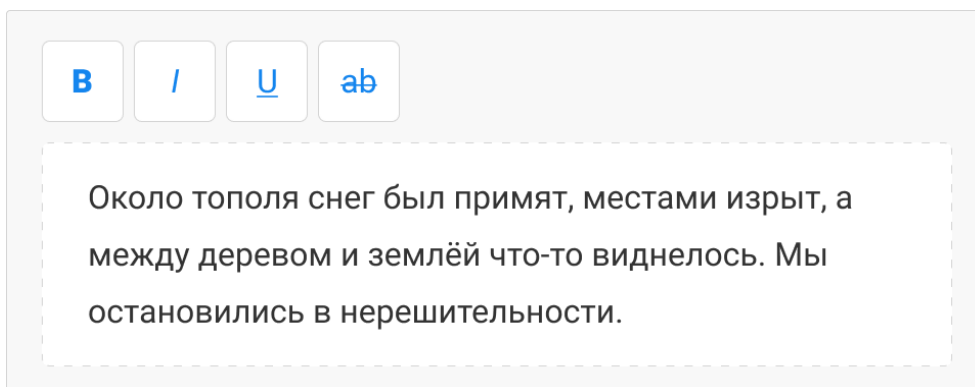
  if (!button) {
    return;
  }

  const setting = button.dataset.setting;

  // переключаем класс, который соответствует кнопке
  text.classList.toggle(setting);
});

```

Теперь наш код не зависит от количества кнопок для форматирования текста. Чтобы проверить это, добавим ещё одну кнопку для форматирования, например, «Зачёркнутый текст».



Кнопка «Зачёркнутый текст»

Добавляем разметку и стили.

```

<section class="editor">
  <h2 class="visually-hidden">Редактор</h2>
  <ul class="editor-toolbar toolbar js-buttons-container">
    <li>
      <button type="button" data-setting="bold" class="toolbar-button

```

```

toolbar-button-bold">B</button>
  </li>
  <li>
    <button type="button" data-setting="italic" class="toolbar-button
toolbar-button-italic">I</button>
  </li>
  <li>
    <button type="button" data-setting="underline" class="toolbar-button
toolbar-button-underline">U</button>
  </li>
  <li>
    <button type="button" data-setting="strikethrough" class="toolbar-button
toolbar-button-strikethrough">ab</button>
  </li>
</ul>
<p class="text">Около тополя снег был примят, местами изрыт, а между деревом и
землей что-то виднелось. Мы остановились в нерешительности.</p>
</section>

```

```

...
.text {
  width: 500px;
  padding: 20px;
  font-size: 24px;
  line-height: 1.5;
  border: 1px solid #421eac;
}

.bold {
  font-weight: 600;
}

.italic {
  font-style: italic;
}

.underline {
  text-decoration: underline;
}

.strikethrough {
  text-decoration: line-through;
}

```

Попробуем применить новый стиль к тексту. Код мы не меняли (дополнили лишь разметку и стили), но всё работает.

Поработаем немного над оформлением кода. Вынесем обработчик события в отдельную функцию и используем деструктурирующее присваивание.

```

// находим панель с кнопками
const toolbar = document.querySelector('.js-buttons-container');

// находим текст
const text = document.querySelector('.text');

const toolbarButtonClick = (evt) => {
  const button = evt.target.closest('button');

  if (!button) {
    return;
  }
}

```

```

const {setting} = button.dataset;

// переключаем класс, который соответствует кнопке
text.classList.toggle(setting);
}

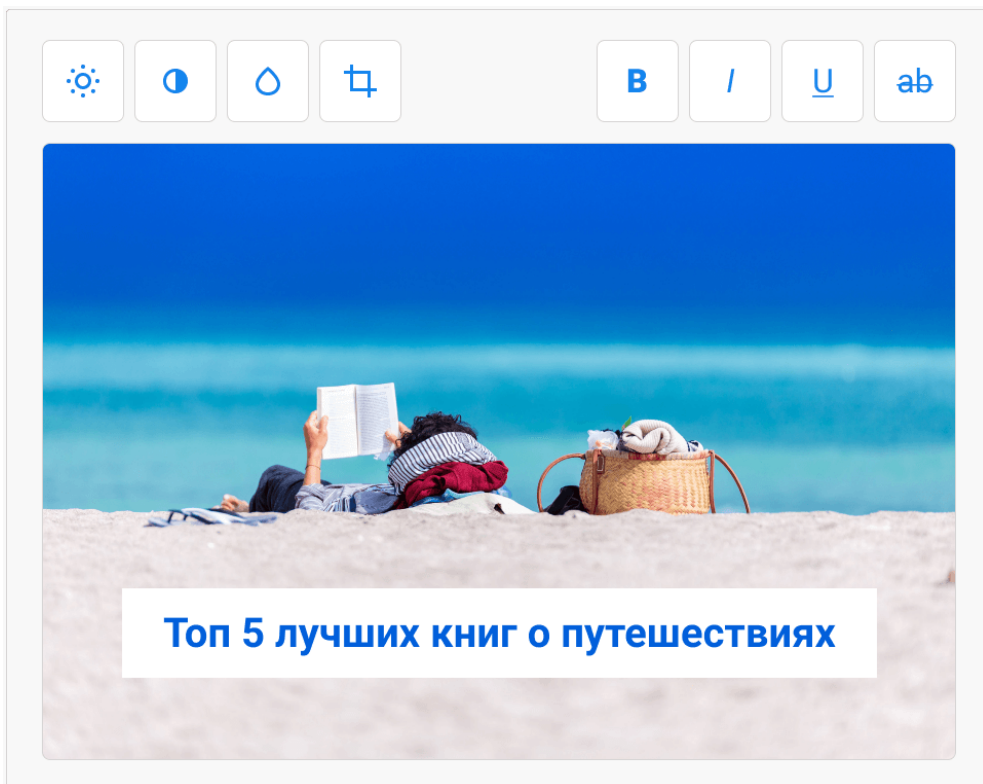
// определяем обработчики событий для кнопок
toolbar.addEventListener('click', toolbarButtonClick);

```

Деструктурирующее присваивание и другие возможности современного синтаксиса JavaScript мы разбираем в материале «[Дополнительные возможности синтаксиса JavaScript](#)».

В процессе уточнения технического задания выясняется, что редактор умеет работать и с изображением. К примеру, к изображению можно применить один из предлагаемых фильтров.

Теперь у нас два элемента для обработки, и у каждого свой набор управляющих кнопок.



Включаем дополнительные возможности по обработке изображений в редактор

Начнём с разметки. Добавим новые элементы:

```

<section class="editor">
  <h2 class="visually-hidden">Редактор</h2>
  <ul class="editor-toolbar toolbar js-buttons-container">
    <li>
      <button class="toolbar-button toolbar-button-brightness"
data-setting="brightness" type="button">Яркость</button>
    </li>
    <li>
      <button class="toolbar-button toolbar-button-contrast"
data-setting="contrast" type="button">Контраст</button>
    </li>
    <li>
      <button class="toolbar-button toolbar-button-blur" data-setting="blur"
type="button">Размытие</button>
    </li>
  </ul>

```

```

    <li>
      <button class="toolbar-button toolbar-button-crop" data-setting="crop"
type="button">Кадрирование</button>
    </li>
    <li>
      <button class="toolbar-button toolbar-button-bold" data-setting="bold"
type="button">B</button>
    </li>
    <li>
      <button class="toolbar-button toolbar-button-italic" data-setting="italic"
type="button">I</button>
    </li>
    <li>
      <button class="toolbar-button
toolbar-button-underline" data-setting="underline" type="button">U</button>
    </li>
    <li>
      <button class="toolbar-button toolbar-button-strikethrough"
data-setting="strikethrough" type="button">ab</button>
    </li>
  </ul>
  <div class="editor-content">
    
    <p class="text">Топ 5 лучших книг о путешествиях</p>
  </div>
</section>

```

По аналогии с функцией для кнопки попробуем хранить целевой элемент в data-атрибутах. Очевидно, что одного data-атрибута нам уже будет недостаточно. Разные кнопки воздействуют на разные элементы, например, перечеркнутый текст может быть применен только тексту, а фильтр Контраст — только к изображению. data-setting переименуем в data-setting-cmd и добавим data-setting-target. В атрибут data-setting-target запишем селектор, с помощью которого можно выбрать целевой элемент. Кстати, селектор :root выбирает элемент, который соответствует тегу <html>.

Итак, разметка с новыми атрибутами:

```

<section class="editor">
  <h2 class="visually-hidden">Редактор</h2>
  <ul class="editor-toolbar toolbar js-buttons-container">
    <li>
      <button class="toolbar-button toolbar-button-brightness"
data-setting-target=".img" data-setting-cmd="brightness"
type="button">Яркость</button>
    </li>
    <li>
      <button class="toolbar-button toolbar-button-contrast"
data-setting-target=".img" data-setting-cmd="contrast"
type="button">Контраст</button>
    </li>
    <li>
      <button class="toolbar-button toolbar-button-blur"
data-setting-target=".img" data-setting-cmd="blur"
type="button">Размытие</button>
    </li>
    <li>
      <button class="toolbar-button toolbar-button-crop"
data-setting-target=".img" data-setting-cmd="crop"
type="button">Кадрирование</button>
    </li>
  </ul>

```

```

    </li>
    <li>
      <button class="toolbar-button toolbar-button-bold"
data-setting-target=".text" data-setting-cmd="bold" type="button">B</button>
    </li>
    <li>
      <button class="toolbar-button toolbar-button-italic"
data-setting-target=".text" data-setting-cmd="italic" type="button">I</button>
    </li>
    <li>
      <button class="toolbar-button toolbar-button-underline"
data-setting-target=".text" data-setting-cmd="underline" type="button">U</button>
    </li>
    <li>
      <button class="toolbar-button toolbar-button-strikethrough"
data-setting-target=".text" data-setting-cmd="strikethrough"
type="button">ab</button>
    </li>
  </ul>
  <div class="editor-content">
    
    <p class="text">Топ 5 лучших книг о путешествиях</p>
  </div>
</section>

```

Доработаем код с учетом изменений.

```

// находим панель с кнопками
const toolbar = document.querySelector('.js-buttons-container');

// определяем обработчики событий для кнопок
toolbar.addEventListener('click', (evt) => {
  const button = evt.target.closest('button');

  if (!button) {
    return;
  }

  const {settingCmd, settingTarget} = button.dataset;

  const element = document.querySelector(settingTarget);
  if (element) {
    // переключаем класс, который соответствует кнопке, на целевом элементе
    element.classList.toggle(settingCmd);
  }
});

```

Теперь код готов, не только к новым управляющим кнопкам, но и к новым элементам, на которые эти кнопки воздействуют.

Пример 15.1. Навигация по сайту библиотеки современных авторов

Техническое задание

О проекте

Библиотека современных авторов — это типовой сайт каталога. Есть алфавитный указатель авторов, группировка книг по автору. В области просмотра книг выбранного автора отображается подробная информация по каждой книге.

Входные данные

Предполагается, что данные для каталога приходят с сервера. Это массив объектов следующего вида:

```
[
  {
    key: 'A',
    items: [
      {
        author: 'Адамс Гай',
        books: [
          {
            genre: 'Детские книги',
            title: 'Очень добрая история',
            year: '2020',
            img: 'book-cover-345.jpg',
            price: '560 ₺',
            summary: 'Мудрая и добрая сказка, в которой просто и проникновенно говорится о самом важном: о дружбе, любви, о том, как важно помогать друг другу. Книга знакомит читателя с историей девочки Анны, которая приехала в маленький угрюмый городок. Своими поступками она запустила цепочку добрых дел и подружила всех жителей города, которые уже и забыли, как радоваться жизни.',
          },
          {
            genre: 'Детские книги',
            title: 'Чудище по имени Лень',
            year: '2019',
            img: 'book-cover-348.jpg',
            price: '480 ₺',
            summary: 'Сказка о необычном чудовище, которое иногда приходит ко всем. Герои сказки узнают, что это самая обычная лень, которая живёт в каждом из нас. На борьбу с ленью они приглашают звёздного мага, который способен победить чудовище.',
          },
          ...
        ],
      },
      {
        author: 'Агутин Леонид',
        books: [
          {
            genre: 'Детские книги',
            title: 'Лесной оркестр',
            year: '2019',
            img: 'book-cover-346.jpg',
            price: '1100 ₺',
            summary: 'Все жители леса объединились, чтобы устроить Мишке сюрприз на День Рождения. Что же было подарком на День Рождения и как отметили этот день лесные жители.',
          },
          {
            genre: 'Художественная литература',
            title: 'Хранитель снов',
            year: '2016',
            img: 'book-cover-353.jpg',
            price: '460 ₺',
            summary: 'Происходит ужасное – у всех людей в мире пропали добрые сны и мечты. На помощь приходит хранитель снов. Но ему не обойтись без помощи своих юных друзей.',
          },
          ...
        ],
      },
    ],
  },
  ...
]
```



```

    },
    ...
  ],
},
{
  key: 'Б',
  items: [
    {
      author: 'Белицкая Светлана',
      books: [
        {
          genre: 'Хобби и досуг',
          title: 'Беззаботная лама',
          year: '2017',
          img: 'book-cover-350.jpg',
          price: '460 ₺',
          summary: 'Легкая и юмористическая история о ламе, которую ничего
никогда не заботит. У нее все всегда хорошо. Ничто на свете не может огорчить
беззаботную ламу.',
        },
        ...
      ],
    },
    ...
  ],
},
{
  key: 'В',
  items: [],
},
...
]

```

key — это буква алфавитного указателя,

items — список авторов, имя которых начинается на букву *key*.

Отдельная запись об авторе состоит из полей:

author — имя автора,

books — список книг автора.

Отдельная запись о книге содержит поля:

genre — жанр,

title — заголовок,

year — год издания,

img — изображение обложки книги,

price — цена,

summary — аннотация к книге.

Описание функциональности

Алфавитный указатель авторов содержит все буквы алфавита и позволяет быстро прокрутить сайт к списку авторов на выбранную букву. Буквы, которые содержат пустой список авторов (`items.length === 0`), визуально отличаются и становятся некликабельными.

Поиск авторов по алфавиту

А Б В Г Д Е Ё Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ъ Э Ю Я

Алфавитный указатель авторов

В списке авторов, помимо имени автора, отображается количество его книг в правильной числовой форме. Например, **11 книг**, **2 книги**, **31 книга**.

А

Абрагимов Руслан
12 книг

Аверин Святослав
2 книги

Адлер Геворк
10 книг

Абрамова Валентина
6 книг

Аврелий Август
10 книг

Азарова Александра
8 книг

Список авторов

По умолчанию в области просмотра книг выводится текст с подсказкой **Нажмите на автора, чтобы посмотреть список его книг**.

Поиск авторов по алфавиту

А Б В Г Д Е Ё Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ъ Э Ю Я

А

Абрагимов Руслан
12 книг

Агутин Леонид
2 книги

Абрамова Валентина
6 книг

Адамс Гай
10 книг

Книги

Нажмите на автора, чтобы посмотреть список его книг

Область просмотра книг. Состояние по умолчанию

При клике на автора область заполняется списком книг этого автора. В списке содержится подробная информация по каждой книге: название, обложка, год издания, цена и аннотация.

Поиск авторов по алфавиту

А Б В Г Д Е Ё Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ъ Ы Э Ю Я

А

Абрагимов Руслан
12 книг

Абрамова Валентина
6 книг

Абрамс Рей
5 книг

Авесян Маринэ
7 книг

Авдеев Петр
7 книг

Аверин Святослав
2 книги

Аврелий Август
10 книг

Авченко Семен
8 книг

Агутин Леонид
2 книги

Адамс Гай
10 книг

Адлер Геворк
10 книг

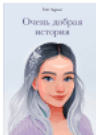
Азарова Александра
8 книг

Айзенберг Евгений
8 книг

Аксенова Кристина
7 книг

Аллилуев Федор
3 книги

Адамс Гай



Очень добрая история

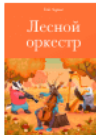
2020

560 ₽

Купить

Мудрая и добрая сказка, в которой просто и проникновенно говорится о самом важном: о дружбе, любви...

Читать дальше



Лесной оркестр


2019

1100 ₽

Купить

Все жители леса объединились, чтобы устроить Мишке сюрприз на День Рождения. Что же было...

Читать дальше



Чудище по имени Лень


2019

480 ₽

Купить

Сказка о необычном чудовище, которое иногда приходит ко всем. Герои сказки узнают, что это...

Читать дальше



Загадка таинственного озера

2018

630 ₽

Купить

Тайну этого озера не могут разгадать уже много лет. Но оно совсем не страшное, а наоборот...

Читать дальше

Область просмотра книг выбранного автора

Аннотация к книге отображается в двух видах: сокращенном и полном. В сокращенном варианте область просмотра ограничена тремя строками и появляется ссылка [Читать дальше](#). При полном варианте отображении текст аннотации показан полностью, а ссылка меняется на [Скрыть описание](#). Переключение между сокращенным и полным вариантом происходит кликом по этой ссылке.

Поиск авторов по алфавиту

А Б В Г Д Е Ё Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ъ Ы Э Ю Я

А

Абрагимов Руслан

12 книг

Абрамова Валентина

6 книг

Абрамс Рей

5 книг

Авесян Маринэ

7 книг

Авдеев Петр

7 книг

Аверин Святослав

2 книги

Аврелий Август

10 книг

Авченко Семен

8 книг

Агутин Леонид

2 книги

Адамс Гай

10 книг

Адлер Геворк

10 книг

Азарова Александра

8 книг

Айзенберг Евгений

8 книг

Аксенова Кристина

7 книг

Аллилуев Федор

3 книги

Адамс Гай

Очень добрая история

2020

560 Р

Купить

Мудрая и добрая сказка, в которой просто и проникновенно говорится о самом важном: о дружбе, любви...

Читать дальше

Лесной оркестр

2019

1100 Р

Купить

Все жители леса объединились, чтобы устроить Мишке сюрприз на День Рождения. Что же было...

Читать дальше

Чудище по имени Лень

2019

480 Р

Купить

Сказка о необычном чудовище, которое иногда приходит ко всем. Герои сказки узнают, что это...

Читать дальше

Загадка таинственного озера

2018

630 Р

Купить

Тайну этого озера не могут разгадать уже много лет. Но оно совсем не страшное, а наоборот...

Читать дальше

Сокращённый вариант отображения аннотации к книге

Поиск авторов по алфавиту

А Б В Г Д Е Ё Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ъ Ы Э Ю Я

А

Абрагимов Руслан

12 книг

Абрамова Валентина

6 книг

Абрамс Рей

5 книг

Авесян Маринэ

7 книг

Авдеев Петр

7 книг

Аверин Святослав

2 книги

Аврелий Август

10 книг

Авченко Семен

8 книг

Агутин Леонид

2 книги

Адамс Гай

10 книг

Адлер Геворк

10 книг

Азарова Александра

8 книг

Айзенберг Евгений

8 книг

Аксенова Кристина

7 книг

Аллилуев Федор

3 книги

Адамс Гай

Очень добрая история

2020

560 Р

Купить

Мудрая и добрая сказка, в которой просто и проникновенно говорится о самом важном: о дружбе, любви, о том, как важно помогать друг другу. Книга знакомит читателя с историей девочки Анны, которая приехала в маленький угрюмый городок. Своими поступками она запустила цепочку добрых дел и подружила всех жителей города, которые уже и забыли, как радоваться жизни.

Скрыть описание

Лесной оркестр

2019

1100 Р

Купить

Все жители леса объединились, чтобы устроить Мишке сюрприз на День Рождения. Что же было...

Читать дальше

Чудище по имени Лень

2019

630 Р

Купить

Тайну этого озера не могут разгадать уже много лет. Но оно совсем не страшное, а наоборот...

Читать дальше

Вариант отображения с полным текстом аннотации к книге

Имя автора, чьи книги находятся в области просмотра, визуально

отличается от других.

В макете содержится вариант с пустым список книг и вариант с заполненным списком книг выбранного автора.

[Скачать макет кейса в формате Figma](#)

Исходные материалы

В кейс включена готовая вёрстка сайта. Структура папок и файлов у сайта следующая:

- в файле `index.html` находится разметка сайта (тут же можно найти разметку для *DOM*-элементов, которые будут добавлены на сайт);
- в папках `css`, `fonts` и `img` находятся необходимые для оформления статические ресурсы;
- в папке `js` есть заготовка для скриптов — файл `script.js` — работу нужно вести в нём.

В файл `script.js` добавлены тестовые (*mock*) входные данные, которые можно использовать для проверки работоспособности кода.

По [ссылке](#) можно скачать стартовый шаблон с версткой страницы.

Теперь разберем, как добавить *DOM*-элементы на веб-страницу и изменить их свойства в *JavaScript*-коде.

1. Первый шагом добавим вёрстку, которую скачаем со стартовым шаблоном.
2. Откроем файл `script.js`. Вместе со стартовым шаблоном пришёл массив данных **authors**. Эти данные используем, чтобы проверить наш код не отправляя запрос на сервер.

/ Тестовые данные */*

```
const authors = [
  {
    key: 'A',
    items: [
      {
        author: 'Адамс Гай',
        books: [
          {
            genre: 'Детские книги',
            title: 'Очень добрая история',
            year: '2020',
            img: 'book-cover-345.jpg',
            price: '560 ₺',
            summary: 'Мудрая и добрая сказка, в которой просто и проникновенно говорится о самом важном: о дружбе, любви, о том, как важно помогать друг другу. Книга знакомит читателя с историей девочки Анны, которая приехала в маленький угрюмый городок. Своими поступками она запустила цепочку добрый дел и подружила всех жителей города, которые уже и забыли, как радоваться жизни.',
          },
          {
            genre: 'Детские книги',
            title: 'Чудище по имени Лень',
            year: '2019',
            img: 'book-cover-348.jpg',
            price: '480 ₺',
            summary: 'Сказка о необычном чудовище, которое иногда приходит ко всем.
```

Герои сказки узнают, что это самая обычная лень, которая живёт в каждом из нас. На борьбу с ленью они приглашают звёздного мага, который способен победить чудовище.',

```

    },
    {
      genre: 'Детские книги',
      title: 'Загадка таинственного озера',
      year: '2018',
      img: 'book-cover-347.jpg',
      price: '630 ₺',
      summary: 'Тайну этого озера не могут разгадать уже много лет. Но оно совсем
не страшное, а наоборот, с волшебной и чистой водой, способной дать силу и уставшему
путнику и раненому зверю.',
    },
  ],
},
{
  author: 'Агутин Леонид',
  books: [
    {
      genre: 'Детские книги',
      title: 'Лесной оркестр',
      year: '2019',
      img: 'book-cover-346.jpg',
      price: '1100 ₺',
      summary: 'Все жители леса объединились, чтобы устроить Мишке сюрприз на
День Рождения. Что же было подарком на День Рождения и как отметили этот день лесные
жители.',
    },
    {
      genre: 'Художественная литература',
      title: 'Хранитель снов',
      year: '2016',
      img: 'book-cover-353.jpg',
      price: '460 ₺',
      summary: 'Происходит ужасное – у всех людей в мире пропали хорошие сны и
мечты. На помощь приходит хранитель снов. Но ему не обойтись без помощи своих юных
друзей.',
    },
  ],
},
],
},
{
  key: 'Б',
  items: [
    {
      author: 'Белицкая Светлана',
      books: [
        {
          genre: 'Хобби и досуг',
          title: 'Беззаботная лама',
          year: '2017',
          img: 'book-cover-350.jpg',
          price: '460 ₺',
          summary: 'Легкая и юмористическая история о ламе, которую ничего никогда не
заботит. У нее все всегда хорошо. Ничто на свете не может огорчить беззаботную ламу.',
        },
      ],
    },
  ],
},
],
},
{
  key: 'В',
  items: [
    {
      author: 'Вагнер Марк',

```

```

books: [
  {
    genre: 'Художественная литература',
    title: 'Единорожка',
    year: '2015',
    img: 'book-cover-354.jpg',
    price: '760 ₺',
    summary: 'Приключения забавного и неуклюжего единорожки Лилу, который
отправился на поиски друзей. Хорошая и поучительная история, полная забавных
приключений.',
  },
],
},
],
{
  key: 'Г',
  items: [],
},
{
  key: 'Д',
  items: [],
},
{
  key: 'Е',
  items: [],
},
{
  key: 'Ё',
  items: [],
},
{
  key: 'Ж',
  items: [],
},
{
  key: 'И',
  items: [],
},
{
  key: 'Й',
  items: [],
},
{
  key: 'К',
  items: [],
},
{
  key: 'Л',
  items: [],
},
{
  key: 'М',
  items: [],
},
{
  key: 'Н',
  items: [],
},
{
  key: 'О',
  items: [],
},
{
  key: 'П',
  items: [],
},

```



```

    },
    {
      key: 'Р',
      items: [],
    },
    {
      key: 'С',
      items: [],
    },
    {
      key: 'Т',
      items: [],
    },
    {
      key: 'У',
      items: [],
    },
    {
      key: 'Ф',
      items: [],
    },
    {
      key: 'Х',
      items: [],
    },
    {
      key: 'Ц',
      items: [],
    },
    {
      key: 'Ч',
      items: [],
    },
    {
      key: 'Ш',
      items: [],
    },
    {
      key: 'Щ',
      items: [],
    },
    {
      key: 'Ы',
      items: [],
    },
    {
      key: 'Э',
      items: [],
    },
    {
      key: 'Ю',
      items: [],
    },
    {
      key: 'Я',
      items: [],
    },
  ],
];

```

3. Переходим к реализации функционала. В первой строке включим **строгий режим**. Так проще будет выявить ошибки. Добавим определение и вызов функции **init**. Эта функция запускается при загрузке страницы.

```
const init = () => {
};

init();
```

4. Контейнер или целевой элемент для алфавитного указателя — это список `<ul class="site-nav">`. Найдём его с помощью метода `document.querySelector` и запишем в переменную `navContainer`. Добавим описание функции `render`, внутри которого очистим содержимое списка. Для очистки используем свойство `innerHTML`. Вызов функции `render` поместим внутрь `init`.

```
const navContainer = document.querySelector('.site-nav');

const render = () => {
  navContainer.innerHTML = '';
}

const init = () => {
  render();
};

init();
```

5. Создадим элементы списка. Для этого используем шаблонные строки и метод `insertAdjacentHTML`. С помощью HTML проще добавить новые элементы на страницу. В разметке можно сразу указать все теги для элементов, включая вложенные, добавить атрибуты для них. Определим функцию `createNavItemString`, в которой соберём строку с HTML для отдельного элемента навигации. Шаблон возьмём в разметке. В параметрах функции передадим ключ элемента и список авторов. У ссылки будем добавлять атрибут `href`, если в списке авторов есть хотя бы один элемент (автор на эту букву). На основе массива данных `authors` создадим новый массив с разметкой для элементов. Для этого используем метод `map`, в котором будем вызывать функцию `createNavItemString`. А затем соберём всю разметку в одну строку с помощью метода `join('')`. Результат запишем в переменную `navElementsString`.

```
const navContainer = document.querySelector('.site-nav');

const createNavItemString = ({key, items}) =>
  `<li class="site-nav-item">
    <a class="site-nav-link" ${items.length > 0 ? `href="#${key}"` :
  ''} >${key}</a>
  </li>
  `;

const render = () => {
  navContainer.innerHTML = '';
```

```

    const navElementsString = authors.map((item) =>
createNavItemString(item)).join('');
}

```

```

const init = () => {
  render();
};

```

6. Добавляем собранный HTML перед закрывающим тегом navContainer. Для этого в первом параметре метода insertAdjacentHTML укажем beforeend. После этого элементы списка появились в DOM.

```

const render = () => {
  navContainer.innerHTML = '';

  const navElementsString = authors.map((item) =>
createNavItemString(item)).join('');

  navContainer.insertAdjacentHTML('beforeend', navElementsString);
}

```

7. Добавим элементы, которые соответствуют разделам с авторами на указанную букву. Целевым элементом этого блока будет <section class="authors">. Найдём этот элемент и затем очистим его содержимое.

```

const navContainer = document.querySelector('.site-nav');
const authorsContainer = document.querySelector('.authors');

const createNavItemString = ({key, items}) =>
`<li class="site-nav-item">
  <a class="site-nav-link" ${items.length > 0 ? `href="#${key}"` :
''} >${key}</a>
  </li>
`;

```

8. Добавим разделы. Для отдельного раздела снова используем HTML из вёрстки. Создаём функцию createSectionString. На вход подаем ключ раздела и список авторов. Ещё раз проходим по массиву данных authors, на этот раз с помощью createSectionString сформируем HTML для всех разделов. Сразу добавим новые элементы в DOM.

```

const createSectionString = ({key, items}) =>
`<section class="authors-section">
  <h2 id="${key}">${key}</h2>
  <ul class="author-list">
    ${items.map(({author, books}) => `
    <li class="author-item">
      <a class="author-link" data-value="${author}" href="#">
        <h3>${author}</h3>
        <p>${books.length} книги</p>
      </a>
    </li>
    `).join('')}
  </ul>
</section>
`;

```

9. Если вы посмотрите на список авторов, скажем, на букву Б, то заметите, что слово книги используется в неверной форме. Должно быть: 1 книга, 2 книги, 5 книг. Для правильного склонения числительных используем внешнюю библиотеку get-declension. Мы выбрали эту библиотеку. Подключаем get-declension в файле index.html с помощью тега `<script>`.

```
</footer>
<script src="js/vendor/get-declension.min.js"></script>
<script src="script.js"></script>
</body>
```

10. На странице с описанием библиотеки в разделе Использование есть примеры вызова функции. Доработаем код, который выводит раздел с авторами, добавим склонение числительных.

```
const createSectionString = ({key, items}) =>
const createSectionString = ({key, items}) =>
`<section class="authors-section">
  <h2 id="${key}">${key}</h2>
  <ul class="author-list">
    ${items.map(({author, books}) => {
      const declension = window.getDeclension({count: books.length, one:
'книга', few: 'книги', many: 'книг'});
      return `
        <li class="author-item">
          <a class="author-link" data-value="${author}" href="#">
            <h3>${author}</h3>
            <p>${declension}</p>
          </a>
        </li>
      `;
    })}.join('')}
  </ul>
</section>
`;
```

11. Далее нам нужно заполнить область с книгами выбранного автора `<section class="books">`. Для начала найдем элемент, который соответствует разделу books. Очистим его содержимое.

```
const navContainer = document.querySelector('.site-nav');
const authorsContainer = document.querySelector('.authors');
const booksContainer = document.querySelector('.books');
---
const authorsElementsString = authors.map((item) =>
createSectionString(item)).join('');

authorsContainer.insertAdjacentHTML('beforeend',
authorsElementsString);

booksContainer.innerHTML = '';
}
```

```
const init = () => {
  render();
};
```

12. По умолчанию, когда автор еще не выбран, в этой области выводится текст с подсказкой Нажмите на автора, чтобы посмотреть список его книг. Определим функцию createEmptyBookString. В ней создадим HTML с заголовком и текстом для раздела. И добавим эти элементы в DOM.

```
....
}).join('')
  </ul>
  </section>
  `;

const createEmptyBookString = () =>
  `<h2>Книги</h2>
  <p>Нажмите на автора, чтобы посмотреть список его книг</p>
  `;
```

```
const render = () => {
  navContainer.innerHTML = '';
  const authorsElementsString = authors.map((item) =>
    createSectionString(item)).join('');
```

```
  authorsContainer.insertAdjacentHTML('beforeend',
    authorsElementsString);
```

```
  booksContainer.innerHTML = '';
```

```
  const emptyBookElementString = createEmptyBookString();
```

```
  booksContainer.insertAdjacentHTML('beforeend',
    emptyBookElementString);
}
```

```
const init = () => {
  render();
};
```

.....

13. Область с книгами должна перерисовываться при клике на автора. Нам понадобится обработчик события click. Чтобы не создавать множество обработчиков событий, используем делегирование и повесим обработчик click на весь элемент authorsContainer. Создадим заготовку функции authorClickHandler. За счет метода closest клик будем отлавливать только на ссылке. В обработчике события отменим поведение ссылки по умолчанию.

```
...
const authorClickHandler = (evt) => {
  const element = evt.target.closest('a');
```

```

    if (!element) {
      return;
    }
    evt.preventDefault();
  };

  const init = () => {
    render();
    authorsContainer.addEventListener('click', authorClickHandler);
  };

  init();

```

14. Продолжим реализацию. Нам потребуется функция для формирования HTML для списка книг автора `createAuthorBooksString`. На вход функция будет принимать объект, который содержит поля `author` и `books`. Разметку для элементов возьмем в вёрстке.

```

...
const createEmptyBookString = () =>
  `<h2>Книги</h2>
  <p>Нажмите на автора, чтобы посмотреть список его книг</p>
  `;

const createAuthorBooksString = ({author, books}) =>
  `<h2>${author}</h2>
  <ul class="book-list">
    ${books.map(({title, year, img, price, summary}) => `
    <li class="author-item">
      <article class="book">
        
        <div class="title-wrapper">
          <h3>${title}</h3>
          <p class="year">${year}</p>
        </div>
        <p class="price">${price} <a class="cart"
href="#">Купить</a></p>
        <p class="description short">${summary}</p>
        <a class="more" href="#">Читать дальше</a>
      </article>
    </li>
    `).join('')}
  </ul>
  `;

const render = () => {
  navContainer.innerHTML = '';
  ...

```

15. При формировании раздела с авторами в атрибуте `data-value` мы добавили имя автора. Используем эти данные для поиска. Сначала найдём группу авторов на выбранную букву: `authors.find(({key}) =>`

value.toUpperCase().indexOf(key.toUpperCase()) === 0). А затем в этой группе отыщем нашего автора: authorsGroup.items.find((item) => item.author === value). Полученную запись будем передавать параметром в функцию createAuthorBooksString. Дополнительные проверки помогут избежать ошибок при некорректных данных. Сформированный *HTML* добавляем в booksContainer. Протестируем наш код, нажмём на несколько ссылок с именами авторов. Область для просмотра книг меняется при смене автора.

```
...
evt.preventDefault();

const value = element.dataset.value;

const authorsGroup = authors.find(({key}) =>
value.toUpperCase().indexOf(key.toUpperCase()) === 0);

if (authorsGroup && authorsGroup.items) {
  const currentAuthor = authorsGroup.items.find((item) => item.author
=== value);

  if (currentAuthor) {
    const booksString = createAuthorBooksString(currentAuthor);

    booksContainer.innerHTML = '';
    booksContainer.insertAdjacentHTML('beforeend', booksString);
  }
}
};
```

```
const init = () => {
  render();
...
}
```

16. Автор, книги которого выводятся в области просмотра, внешне отличается от остальных, у ссылки появляется дополнительный класс active. Создадим функцию SetAuthorActive, она будет снимать класс active у элемента, который был до этого активным и добавлять этот класс на элемент, по которому кликнули.

```
...

const setAuthorActive = (target) => {
  const activeAuthor =
authorsContainer.querySelector('.author-link.active');
  if (activeAuthor) {
    activeAuthor.classList.remove('active');
  }

  target.classList.add('active');
};

const authorClickHandler = (evt) => {
...
...
  if (currentAuthor) {
```



```

    const booksString = createAuthorBooksString(currentAuthor);

    booksContainer.innerHTML = '';
    booksContainer.insertAdjacentHTML('beforeend', booksString);
    setAuthorActive(element);
  }
}
};

const init = () => {
  render();

```

17. В техническом задании написано, что аннотация к книге по умолчанию отображается в сокращенном виде, а при клике на ссылку Читать дальше показывается полностью. Реализуем этот функционал. Определяем обработчик события click для области просмотра книг. Снова воспользуемся делегированием. На этот раз событие будем «ловить» на ссылках с классом more.

```

...
const moreLinkClickHandler = (evt) => {
  const element = evt.target.closest('a.more');

  if (!element) {
    return;
  }
  evt.preventDefault();
};

const init = () => {
  render();

  authorsContainer.addEventListener('click', authorClickHandler);
  booksContainer.addEventListener('click', moreLinkClickHandler);
};

init();

```

18. Нам понадобится элемент с классом description, для него будем переключать класс short. Находим элемент с помощью метода querySelector. Метод вызываем для bookContainer (ближайший родитель с классом book для ссылки, на которой произошло событие). Тут же будем менять текстовое содержимое самой ссылки. Если описание содержит класс short, то в ссылке показывается текст Читать дальше, в противном случае — Скрыть описание. Для замены текстового содержимого используем свойство textContent.

```

....

const bookContainer = element.closest('.book');
const descriptionElement =
bookContainer.querySelector('.description');

```

```

if (descriptionElement) {
    descriptionElement.classList.toggle('short');
    element.textContent =
descriptionElement.classList.contains('short') ? 'Читать дальше' :
'Скрыть описание';
}
};

```

19. Следующим шагом реализуем плавную прокрутку. Для начала добавим обработчик события click для контейнера с навигацией. Отлавливать событие будем на ссылках. Для ссылок отменим действие по умолчанию.

.....

```

const navItemsClickHandler = (evt) => {
    const element = evt.target.closest('a');

    if (!element) {
        return;
    }
    evt.preventDefault();
};

```

20. Ссылки с буквами в алфавитном указателе содержат в значении атрибута href указатели на якоря соответствующих разделов. Используем значение атрибута href, чтобы найти нужный элемент и вызвать для него метод scrollToView. Реализацию плавной прокрутки мы разбираем в методике (см. выше по тексту).

...

```

evt.preventDefault();

const blockId = element.getAttribute('href');

if (blockId && blockId !== '#') {
    const block = document.querySelector(blockId);

    if (block) {
        block.scrollToView({
            behavior: 'smooth',
            block: 'start',
        });
    }
}
};

```

21. Весь необходимый функционал реализован. Но наш код не идеален. Есть много повторяющихся фрагментов, например, у нас несколько раз повторяется последовательность команд: очистить содержимое родительского элемента, добавить HTML в родительский элемент. Выделим эти действия в отдельную функцию addChildElementsFromString. На вход будем подавать родительский элемент и HTML для дочерних элементы. Пройдёмся по коду и заменим повторяющиеся команды на

ВЫЗОВ НОВОЙ ФУНКЦИИ.

....

```
const addChildElementsFromString = (parent, domString) => {  
  parent.innerHTML = '';  
  // добавляем разобранный текст как HTML перед закрывающим тегом  
  родительского элемента (после последнего потомка)  
  parent.insertAdjacentHTML('beforeend', domString);  
};
```

....

22. Немного отрендерим синтаксис. Поскольку функции `createNavItemString` и `createSectionString` не используют данные из контекста (`this`), их можно передать как параметр в метод `map`.

23. Используем деструктурирующее присваивание, оператор `const` `value = element.dataset.value`; заменим на `const {value} = element.dataset`; Деструктурирующее присваивание и другие возможности синтаксиса *JavaScript* мы рассматриваем в разделе [«Дополнительные возможности синтаксиса JavaScript»](#). Обязательно после каждого «улучшения» проверяйте функционал. Сайт библиотеки современных авторов готов!

```
const value = element.dataset.value;  
const {value} = element.dataset;
```

Результат

```
'use strict';
```

```
/* Тестовые данные */
```

```
const authors = [  
  {  
    key: 'A',  
    items: [  
      {  
        author: 'Адамс Гай',  
        books: [  
          {  
            genre: 'Детские книги',  
            title: 'Очень добрая история',  
            year: '2020',  
            img: 'book-cover-345.jpg',  
            price: '560 ₺',  
            summary: 'Мудрая и добрая сказка, в которой просто и
```

```
            summary: 'Мудрая и добрая сказка, в которой просто и  
            проникновенно говорится о самом важном: о дружбе, любви, о том, как  
            важно помогать друг другу. Книга знакомит читателя с историей девочки  
            Анны, которая приехала в маленький угрюмый городок. Своими поступками  
            она запустила цепочку добрых дел и подружила всех жителей города,  
            которые уже и забыли, как радоваться жизни.',  
          },  
        ],  
      },  
    ],  
  },  
];
```

```

{
  genre: 'Детские книги',
  title: 'Чудище по имени Лень',
  year: '2019',
  img: 'book-cover-348.jpg',
  price: '480 ₺',
  summary: 'Сказка о необычном чудовище, которое иногда
приходит ко всем. Герои сказки узнают, что это самая обычная лень,
которая живёт в каждом из нас. На борьбу с ленью они приглашают
звёздного мага, который способен победить чудовище.',
},
{
  genre: 'Детские книги',
  title: 'Загадка таинственного озера',
  year: '2018',
  img: 'book-cover-347.jpg',
  price: '630 ₺',
  summary: 'Тайну этого озера не могут разгадать уже много лет.
Но оно совсем не страшное, а наоборот, с волшебной и чистой водой,
способной дать силу и уставшему путнику и раненому зверю.',
},
],
},
{
  author: 'Агутин Леонид',
  books: [
    {
      genre: 'Детские книги',
      title: 'Лесной оркестр',
      year: '2019',
      img: 'book-cover-346.jpg',
      price: '1100 ₺',
      summary: 'Все жители леса объединились, чтобы устроить
Мишке сюрприз на День Рождения. Что же было подарком на День
Рождения и как отметили этот день лесные жители.',
    },
    {
      genre: 'Художественная литература',
      title: 'Хранитель снов',
      year: '2016',
      img: 'book-cover-353.jpg',
      price: '460 ₺',
      summary: 'Происходит ужасное – у всех людей в мире пропали
добрые сны и мечты. На помощь приходит хранитель снов. Но ему не
обойтись без помощи своих юных друзей.',
    },
  ],
},
],
},
{
  key: 'Б',
  items: [
    {
      author: 'Белицкая Светлана',

```

```

books: [
  {
    genre: 'Хобби и досуг',
    title: 'Беззаботная лама',
    year: '2017',
    img: 'book-cover-350.jpg',
    price: '460 ₺',
    summary: 'Легкая и юмористическая история о ламе, которую
ничего никогда не заботит. У нее все всегда хорошо. Ничто на свете не
может огорчить беззаботную ламу.',
  },
],
},
],
{
  key: 'В',
  items: [
    {
      author: 'Вагнер Марк',
      books: [
        {
          genre: 'Художественная литература',
          title: 'Единорожка',
          year: '2015',
          img: 'book-cover-354.jpg',
          price: '760 ₺',
          summary: 'Приключения забавного и неуклюжего единорожки
Лилу, который отправился на поиски друзей. Добрая и поучительная
история, полная забавных приключений.',
        },
      ],
    },
  ],
},
],
{
  key: 'Г',
  items: [],
},
{
  key: 'Д',
  items: [],
},
{
  key: 'Е',
  items: [],
},
{
  key: 'Ё',
  items: [],
},
{
  key: 'Ж',
  items: [],
},

```

```
{
  key: 'И',
  items: [],
},
{
  key: 'Й',
  items: [],
},
{
  key: 'К',
  items: [],
},
{
  key: 'Л',
  items: [],
},
{
  key: 'М',
  items: [],
},
{
  key: 'Н',
  items: [],
},
{
  key: 'О',
  items: [],
},
{
  key: 'П',
  items: [],
},
{
  key: 'Р',
  items: [],
},
{
  key: 'С',
  items: [],
},
{
  key: 'Т',
  items: [],
},
{
  key: 'У',
  items: [],
},
{
  key: 'Ф',
  items: [],
},
{
  key: 'Х',
  items: [],
}
```

```

    },
    {
      key: 'Ц',
      items: [],
    },
    {
      key: 'Ч',
      items: [],
    },
    {
      key: 'Ш',
      items: [],
    },
    {
      key: 'Щ',
      items: [],
    },
    {
      key: 'Ы',
      items: [],
    },
    {
      key: 'Э',
      items: [],
    },
    {
      key: 'Ю',
      items: [],
    },
    {
      key: 'Я',
      items: [],
    },
  ],
];

```

```

const navContainer = document.querySelector('.site-nav');
const authorsContainer = document.querySelector('.authors');
const booksContainer = document.querySelector('.books');

```

```

const addChildElementsFromString = (parent, domString) => {
  parent.innerHTML = '';
  // добавляем разобранный текст как HTML перед закрывающим тегом
  // родительского элемента (после последнего потомка)
  parent.insertAdjacentHTML('beforeend', domString);
};

```

```

const createNavItemString = ({key, items}) =>
  `<li class="site-nav-item">
    <a class="site-nav-link" ${items.length > 0 ? `href="#${key}"` :
  ''} >${key}</a>
  </li>
  `;

```

```

const createSectionString = ({key, items}) =>
  `<section class="authors-section">

```

```

    <h2 id="${key}">${key}</h2>
    <ul class="author-list">
      ${items.map(({author, books}) => {
const declension = window.getDeclension({count: books.length, one:
'книга', few: 'книги', many: 'книг'}));
return `
        <li class="author-item">
          <a class="author-link" data-value="${author}" href="#">
            <h3>${author}</h3>
            <p>${declension}</p>
          </a>
        </li>
      `;
    }).join('')}
    </ul>
  </section>
`;

const createEmptyBookString = () =>
`<h2>Книги</h2>
  <p>Нажмите на автора, чтобы посмотреть список его книг</p>
`;

const createAuthorBooksString = ({author, books}) =>
`<h2>${author}</h2>
  <ul class="book-list">
    ${books.map(({title, year, img, price, summary}) => `
      <li class="author-item">
        <article class="book">
          
          <div class="title-wrapper">
            <h3>${title}</h3>
            <p class="year">${year}</p>
          </div>
          <p class="price">${price} <a class="cart"
href="#">Купить</a></p>
          <p class="description short">${summary}</p>
          <a class="more" href="#">Читать дальше</a>
        </article>
      </li>
    `).join('')}
  </ul>
`;

const render = () => {
  const navElementsString = authors.map(createNavItemString).join('');
  const authorsElementsString =
authors.map(createSectionString).join('');
  const emptyBookElementString = createEmptyBookString();

  addChildElementsFromString(navContainer, navElementsString);
  addChildElementsFromString(authorsContainer, authorsElementsString);
  addChildElementsFromString(booksContainer, emptyBookElementString);
}

```



```

const setAuthorActive = (target) => {
  const activeAuthor =
authorsContainer.querySelector('.author-link.active');
  if (activeAuthor) {
    activeAuthor.classList.remove('active');
  }

  target.classList.add('active');
};

const authorClickHandler = (evt) => {
  const element = evt.target.closest('a');

  if (!element) {
    return;
  }
  evt.preventDefault();

  const {value} = element.dataset;

  const authorsGroup = authors.find(({key}) =>
value.toUpperCase().indexOf(key.toUpperCase()) === 0);

  if (authorsGroup && authorsGroup.items) {
    const currentAuthor = authorsGroup.items.find((item) => item.author
=== value);

    if (currentAuthor) {
      const booksString = createAuthorBooksString(currentAuthor);

      addChildElementsFromString(booksContainer, booksString);
      setAuthorActive(element);
    }
  }
};

const moreLinkClickHandler = (evt) => {
  const element = evt.target.closest('a.more');

  if (!element) {
    return;
  }
  evt.preventDefault();

  const bookContainer = element.closest('.book');
  const descriptionElement =
bookContainer.querySelector('.description');

  if (descriptionElement) {
    descriptionElement.classList.toggle('short');
    element.textContent =
descriptionElement.classList.contains('short') ? 'Читать дальше' :
'Скрыть описание';
  }
}

```

```

};

const navItemsClickHandler = (evt) => {
  const element = evt.target.closest('a');

  if (!element) {
    return;
  }
  evt.preventDefault();

  const blockId = element.getAttribute('href');

  if (blockId && blockId !== '#') {
    const block = document.querySelector(blockId);

    if (block) {
      block.scrollIntoView({
        behavior: 'smooth',
        block: 'start',
      });
    }
  }
};

const init = () => {
  render();

  authorsContainer.addEventListener('click', authorClickHandler);
  booksContainer.addEventListener('click', moreLinkClickHandler);
  navContainer.addEventListener('click', navItemsClickHandler);
};

init();

```

Задание 15.2. Создайте скрипт для “Оживления” словаря психологических терминов

Техническое задание

О проекте

Словарь основных психологических терминов — это типовой сайт глоссария. На сайте есть алфавитный указатель.

Входные данные

Предполагается, что данные для глоссария приходят с сервера. Это массив объектов следующего вида:

```
[
  {
    key: 'А',
    items: [
      {
        term: 'Абстиненция',
        description: '<p><dfn>Абстинѐнция</dfn> (от <a href="#">нем.</a> <i>Abstinenz</i> и <a href="#">лат.</a> <i>Abstinentia</i>: воздержность, задержка, задержание, воздержание, воздержанность, а также пост, голодание, бескорыстие, честность) – в отличие от традиционного, <a href="#">наркологического</a> значения термина, в <a href="#">психоанализе</a> употребляется для описания состояния больных истерией страха и неврозом навязчивых состояний в процессе психоаналитической терапии. Абстиненция рассматривается не как патогенный фактор, а скорее как фактор, поддерживающий оптимальный уровень <a href="#">фрустрации</a>, необходимый для достижения терапевтического эффекта.</p>',
        cite: 'Материал из Википедии – свободной энциклопедии',
      },
      {
        term: 'Авторитетность',
        description: '<p>Имеющий <a href="#">авторитет</a>, заслуживающий признания, доверия и уважения.</p><p>Как видите, работа даже такого бесспорно авторитетного режиссера, как Станиславский, подвергалась некоторому сомнению со стороны критики. Блейхман отвечал без лишних слов: Совет для анархистов совершенно не авторитетен; если к его решению присоединятся большевики, то это ничего не значит, Совет в целом служит буржуазии и помещикам; никаких определенных намерений у анархистов на завтра нет; участвовать в манифестации они будут — со своими чёрными знамёнами, а насчет того, будут ли с оружием, то, может быть, пойдут без оружия, а может быть, и с оружием.</p>',
        cite: 'Н. Н. Суханов, «Записки о революции / Книга 4», 1918–1921 г. (цитата из Национального корпуса русского языка, см. Список литературы)',
      },
      ...
    ]
  },
  {
    key: 'Б',
    items: [
      {
        term: 'Базальные эмоции',
        description: '<p><dfn>Базальные эмоции</dfn> – теоретический конструкт, объединяющий эмоции минимального набора, на базе которых формируется все многообразие эмоциональных процессов и состояний. К подобным эмоциям относят эмоции радости, горя (печали), страха, гнева, удивления, отвращения.</p>',
        cite: 'Материал из Википедии – свободной энциклопедии',
      },
      {
        term: 'Бионика',
        description: '<p><dfn>Биѐника</dfn> (от <a href="#">др.-греч.</a> βίον «живущее») – прикладная наука о применении в технических устройствах и системах принципов организации, свойств, функций и структур живой природы, то есть формах живого в природе и их промышленных аналогах.</p>',
        cite: 'Материал из Википедии – свободной энциклопедии',
      },
      ...
    ]
  },
]
```

```
{
  key: 'B',
  items: [],
}
...
```

key — это буква алфавитного указателя,

items — список терминов, название которых начинается на букву *key*.

Отдельный термин состоит из полей:

term — название термина,

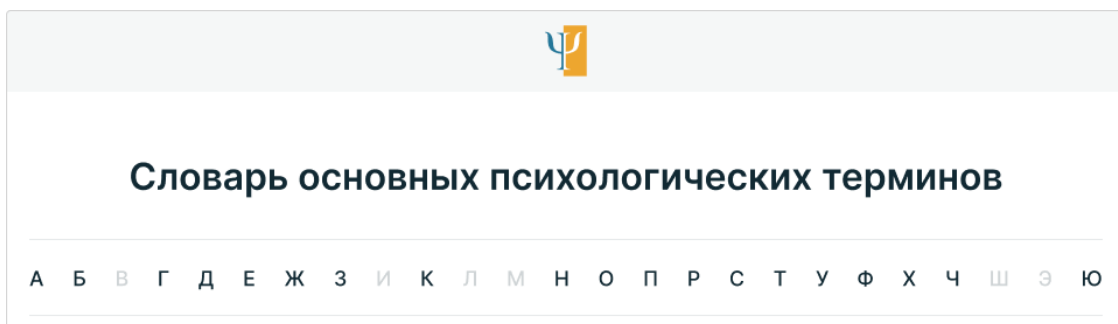
description — описание термина, может содержать HTML,

cite — источник.

Описание функциональности

Алфавитный указатель содержит все буквы алфавита и позволяет быстро прокрутить сайт к списку терминов на выбранную букву.

Буквы, которые содержат пустой список терминов (`items.length === 0`), становятся некликабельными (у ссылок отсутствует атрибут `href`, например, `В`). Также они отличаются визуально.



Алфавитный указатель психологических терминов

Для пустого списка терминов раздел на эту букву не отображается.

А	
Адаптация	Атрибуция
Аддикция	Аффилиация
Б	
Базальные эмоции	Бионика
Г	
Галлюцинации	

За разделом на букву Б идёт раздел на букву Г, раздел на букву В не выводится

При клике на термин открывается попап-окно с полным определением этого психологического термина и источником.

А
Абстиненция
Абулия
Авторитетность
Адаптация
Аддикция

Акцентуация

Абстиненция (от [нем.](#) *Abstinenz* и [лат.](#) *Abstinentia*: воздержность, задержка, задержание, воздержание, воздержанность, а также пост, голодание, бескорыстие, честность) — в отличие от традиционного, [наркологического](#) значения термина, в [психоанализе](#) употребляется для описания состояния больных истерией страха и неврозом навязчивых состояний в процессе психоаналитической терапии. Абстиненция рассматривается не как патогенный фактор, а скорее как фактор, поддерживающий оптимальный уровень [фрустрации](#), необходимый для достижения терапевтического эффекта.

Материал из Википедии — свободной энциклопедии

Попап-окно с полным определением психологического термина

Попап-окно закрывается по клику на кнопку **Заккрыть**.

Абстиненция (от нем. *Abstinenz* и лат. *Abstinencia*: воздержность, задержка, задержание, воздержание, воздержанность, а также пост, голодание, бескорыстие, честность) — в отличие от традиционного, наркологического значения термина, в психоанализе употребляется для описания состояния больных истерией страха и неврозом навязчивых состояний в процессе психоаналитической терапии. Абстиненция рассматривается не как патогенный фактор, а скорее как фактор, поддерживающий оптимальный уровень фрустрации, необходимый для достижения терапевтического эффекта.

Материал из Википедии — свободной энциклопедии

Кнопка **Закреть**.

Определение термина может содержать в себе теги, их следует отображать как есть. В макете показаны оба состояния: попап-окно закрыто и попап-окно открыто.

[Скачать макет кейса в формате Figma](#)

Исходные материалы

В кейс включена готовая вёрстка сайта. Структура папок и файлов у сайта следующая:

в файле **index.html** находится разметка сайта (тут же можно найти разметку для DOM-элементов, которые будут добавлены на сайт);

в папках **css**, **fonts** и **img** находятся необходимые для оформления статические ресурсы;

в папке **js** есть заготовка для скриптов — файл **script.js** — работу нужно вести в нём.

В файл **script.js** добавлены тестовые (*mock*) входные данные, которые можно использовать для проверки работоспособности кода.

По [ссылке](#) можно скачать стартовый шаблон с версткой страницы.

Дополнительная информация

Все необходимые CSS-классы и CSS-правила есть в исходных материалах. Для реализации функционала достаточно добавить

элементам соответствующие классы.

Описание CSS-классов

CSS-класс `shown` — класс для вывода попап-а с определением термина.

Для того, чтобы показать окно, нужно добавить этот класс для элемента

`<div class="modal">` и разметка окна будет выглядеть так:

```
<div class="modal shown">
```

Скачайте [стартовый шаблон](#) навыка и напишите JS-код в файле `js/script.js`.

Инструкция

Чтобы проверить своё решение и довести его до идеала, действуйте так:

1. Выполните задачу кейса.
2. Загрузите файлы в `repl.it` или на `github` для проверки.
3. Сформируйте внешнюю ссылку на деплой проекта.

Каким должен быть архив с версткой?

Сразу внутри папки с проектом должен находиться файл `index.html`.

В папке `js` должен быть `script.js`, подключенный в `index.html`.

Папки и файлы должны содержать только латинские символы. Других ограничений нет.

Вспомогательные материалы

1. https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/JavaScript_basics
2. [https://www.w3schools.com/jquery/default.a](https://www.w3schools.com/jquery/default.asp)
[sp](#)
- 3.