Библиотека JavaScript D3.js

О чем этот документ?

Основные положения библиотеки D3

Что такое d3

D3.js (Data-Driven Documents) – это библиотека на языке JavaScript для обработки и визуализации данных.

Библиотека D3.js основана на использовании JavaScript, SVG и CSS, то есть поддерживает векторную графику. Это позволяет создавать структуры с графикой, обладающие анимацией и возможностями взаимодействия.

Ссылка на библиотеку:

```
<script src="http://d3js.org/d3.v7.js"></script>
```

Пример. Разместим на html страницу прямоугольник с помощью SVG и D3.js.

HTML-код

```
<head>
  <script src="http://d3js.org/d3.v7.min.js"> </script>
</head>
<body>
  <h3>SVG</h3>
  <svg id="svg">
    <rect width="150" height="100" style="fill:grey;">
    </rect>
  </svg>

  <h3>D3.js</h3>
  <svg id="d3">
  </svg>

  <script>
    d3.select("svg#d3")
      .append("rect")
      .attr("width", 150)
      .attr("height", 100)
      .style("fill", "grey");
  </script>
</body>
```

Результат:

В скрипте в svg с **id=d3** формируется такой же html-кодом, что и в svg с **id=svg**. В нем устанавливаются аналогичные атрибуты и стиль.

Выбор элементов

Для выбора определенного элемента из структуры DOM, используется селектор. Селектор представляет собой некий шаблон, которому должны соответствовать элементы. В качестве селекторов применяются css-селекторы.

D3.js для выбора предоставляет два метода:

- **d3.select("selector")** - возвращает первый элемент из всех элементов, которые соответствуют селектору;
- **d3.selectAll("selector")** - возвращает все элементы, которые соответствуют селектору

Пример. Рассмотрим html-страницу:

```
<head>
  <script src="http://d3js.org/d3.v7.min.js"> </script>
</head>
<body>
  <div class="menu">
    <ul>
      <li>Основы JavaScript</li>
      <li>CSS</li>
      <li class="active">Библиотеки JavaScript</li>
    </ul>
  </div>
  <div class="main">
    <p>
      Библиотеки JavaScript:
    </p>
    <ol>
      <li class="red">D3</li>
      <li class="green">JQuery</li>
      <li class="red">Chart</li>
    </ol>
  </div>
</body>
```

Выбор элементов с помощью d3:

- **d3.selectAll("li")** – все элементы, относящиеся к тегу **li**
- **d3.select ("li")** – первый элемент с тегом **li** (**Основы JavaScript**)
- **d3.selectAll("div.menu li")** – все **li**, являющиеся потомками элемента **div** класса **menu**

Можно использовать несколько уровней выборки:

- `d3.selectAll("div.menu").select("li")` - первый **li**, являющийся потомком элемента `div` класса `menu`

Операторы

Операторы представляют методы, которые позволяют выполнять различные операции над выборкой. Существует несколько основных операторов:

- **attr()** - получает или устанавливает значение атрибута

- **style()** - получает или устанавливает стиль элемента
- **text()** - получает или устанавливает текстовое содержимое
- **html()** - получает или устанавливает html-код элемента

Оператор attr()

Получение значение атрибута:

```
let attrValue = d3.select("ol li").attr("class");
```

Получаем значение атрибута **class** первого **li**, относящегося к нумерованному списку. Результат – **red**.

Установка значения атрибута:

```
d3.selectAll("p").attr("class", "red");
```

Всем абзацам документа присваиваем класс **red**.

Оператор style()

Установка стиля:

```
d3.select("div.menu")
  .style("width", "200px")
  .style("border", "solid thin green");
```

С помощью цепочки вызовов устанавливаем стили для ширины и границы.

Получение стиля:

```
let width = d3.select("div.menu").style("width");
```

Переменной width будет присвоено значение **200px**.

Оператор text()

Получение текста:

```
let textLi = d3.select("ol li").text();
```

Получаем текст первого **li**, относящегося к нумерованному списку. Результат – **D3**.

Установка текста:

```
d3.select("p").text("Популярные библиотеки:");
```

Оператор html()

Установка кода html:

```
d3.select("ol li")
  .html(`<a href=#>${ d3.select("ol li").text()}</a>`);
```

Оборачивает в ссылку текст первого **li**, относящегося к нумерованному списку. Результат - `<li><a href="#">D3</a></li>`

Получение html:

```
let htmlCode = d3.select("div.menu").html();
```

Результат:

```
<ul>
  <li>Основы JavaScript</li>
  <li>CSS</li>
  <li class="active">Библиотеки JavaScript</li>
</ul>
```

Создание и удаление элементов

Для манипулирования элементами D3.js используются следующие операторы:

- **append()** - добавляет новый элемент, тег который передается в качестве параметра;
- **insert()** - добавляет новый элемент перед заданным;
- **remove()** - удаляет элемент.

Оператор append()

Добавление элемента:

```
d3.select("ol").append("li").text("SurveyJS");
```

Оператор **append** в качестве параметра получает название тега. Тег **li** добавляется после последнего дочернего элемента первого встретившегося тега **ol**. Все операторы, которые идут по цепочке после метода **append**, применяются к создаваемому элементу **li**.

Результат:

```
<ol>
  <li class="red">D3</li>
  <li class="green">JQuery</li>
  <li class="red">Chart</li>
  <li>SurveyJS</li>
 </ol>
```

Оператор insert()

Оператор **insert()** позволяет вставить элемент перед заданным:

```
d3.select("ol").insert("li", "li.green").text("SurveyJS");
```

Создаваемый элемент **li** добавляется перед первым найденным в элементе **ol** элементом **li** с классом, относящимся к классу **green**:

Результат:

```
<ol>
  <li class="red">D3</li>
  <li>SurveyJS</li>
  <li class="green">JQuery</li>
  <li class="red">Chart</li>
 </ol>
```

Оператор remove()

Оператор **remove()** удаляет элемент (элементы):

```
d3.select("ol").selectAll("li.red").remove();
```

Удаляет все **li** класса **red**, относящиеся к первому на страницу тегу **ol**.

Работа с данными

Привязка элементов

Для добавления данных в элемент применяется метод **`data()`**. В качестве аргумента в этот метод передается массив объектов.

Пример. Рассмотрим html-страницу:

HTML-код:

```
<head>
  <script src="http://d3js.org/d3.v7.min.js"> </script>
</head>
<body>
  <div class="main">
    <p>
      Библиотеки JavaScript:
    </p>
    <ol>
      <li></li>
      <li></li>
      <li></li>
    </ol>
  </div>
</body>
<script src="program.js"> </script>
```

Сформируем нумерованный список на основе данных из массива

JavaScript-код:

```
let libJS = ['D3', 'JQuery', 'Chart'];
d3.select("ol")
  .selectAll("li")
  .data(libJS)
  .text( function(d){
     return d;
  });
```

Метод **`data()`** устанавливает связь между элементами html и элементами массива. Чтобы выполнить привязку элементов, используется функция динамической модификации (dynamic modifier function).

```
function(d [,i]){
     return d;
  }
```

В функцию передаются два параметра:

`d` – очередной элемент массива;

`i` - индекс элемента (необязательный параметр).

Эта функция вызывается для каждого элемента данных, отобранных методом **`date()`**, и возвращает этот элемент. Элемент данных привязывается к соответствующему элементу, отобранному методом **`selectAll()`**.

Также в качестве функции динамической модификации можно использовать стрелочную функцию.

JavaScript-код:

```
let libJS = ['D3', 'JQuery', 'Chart'];
d3.select("ol")
  .selectAll("li")
  .data(libJS)
  .text((d)=> d);
```

Результат:

```
<ol>
  <li>D3</li>
  <li>JQuery</li>
  <li>Chart</li>
</ol>
```

Привязка элементов с созданием новых

В случае если, количество элементов в массиве данных больше, чем количество элементов, к которым привязывается этот массив, в документе будут отображены столько элементов данных, сколько доступно элементов в html-документе.

Чтобы ввести дополнительные элементы (чтобы отобразить все данные) используется метод **`enter()`**.

JavaScript-код:

```
let libJS = ['D3', 'JQuery', 'Chart', ' SurveyJS ', ' Parsley'];
// создаем необходимое количество html-элементов
  d3.select("ol")
    .selectAll("li")
    .data(libJS)
    .enter()
    .append('li');

//выводим данные
  d3.select("ol")
    .selectAll("li")
    .data(libJS)
    .text(d => d)
```

Результат:

```
<ol>
  <li>D3</li>
  <li>JQuery</li>
  <li>Chart</li>
  <li> SurveyJS </li>
  <li> Parsley </li>
</ol>
```

Привязка элементов с созданием новых и удалением ненужных

Метод **`exit()`** выделяет те html-элементы, которые не привязаны к данным.

Пусть в нашем примере уже сформирован какой-то список, причем неизвестно в нем больше или меньше элементов, чем в массиве данных.

HTML-код:

```
<head>
  <script src="http://d3js.org/d3.v7.min.js"> </script>
</head>
<body>
  <div class="main">
    <p>
      Библиотеки JavaScript:
    </p>
    <ol>
          <li>1</li>
          <li>2</li>
          <li>3</li>
```

```
            <li>4</li>
            <li>5</li>
            <li>6</li>
            <li>7</li>
            <li>8</li>
            <li>9</li>
    </ol>
  </div>
</body>
<script src="program.js"> </script>
```

Тогда в наш скрипт нужно добавить отбор лишних элементов и удаление их.

JavaScript-код:

```
let libJS = ['D3', 'JQuery', 'Chart', ' SurveyJS ', ' Parsley'];
// создаем необходимое количество html-элементов
  d3.select("ol")
    .selectAll("li")
    .data(libJS)
    .enter()
    .append('li');

//выводим данные
  d3.select("ol")
    .selectAll("li")
    .data(libJS)
    .text((d)=> d);

// удаляем лишние html-элементы
  d3.select("ol")
    .selectAll("li")
    .data(libJS)
    .exit()
    .remove();
```

Результат:

```
    <ol>
      <li>D3</li>
      <li>JQuery</li>
      <li>Chart</li>
      <li> SurveyJS </li>
      <li> Parsley </li>
   </ol>
```

Функции работы с данными D3

Функции работы с данными:

- **`d3.min(data)`** - возвращает минимальное значение в массиве **`data;`**
- **`d3.max(data)`** - возвращает максимальное значение;
- **`d3.extent(data)`** - возвращает минимальное и максимальное значения в виде массива из двух элементов;
- **`d3.sum(data)`** - возвращает сумму всех элементов массива;
- **`d3.median(data)`** - возвращает медиану массива;
- **`d3.mean(data)`** - получает среднее значение;
- **`d3.bisect(data, element)`** - функция, которая возвращает позицию в отсортированном массиве, куда можно поместить определенный объект, не нарушая порядка.

• **d3.group()**- функция, которая возвращает сгруппированные по некоторому ключу данные.

Группировка и вычисления в группах

Пример. Рассмотрим массив с описанием книг и для каждого автора выведем максимальную и минимальную цену его книг, а таже суммарную стоимость всех экземпляров книг.

JavaScript-код:

```
let books = [
    {title : 'Мастер и Маргарита',
     author: 'Булгаков М.А.',
     price: 581.50,
     amount: 10
    },
    {title :  'Белая гвардия',
     author: 'Булгаков М.А.',
     price: 600.00,
     amount: 10
    },
    {title : 'Война и мир',
     author: 'Толстой Л.Н.',
     price: 899.99,
     amount: 10
    },
    {title : 'Анна Каренина',
     author: 'Толстой Л.Н.',
     price: 450.10,
     amount: 15
    },
    {title : 'Игрок',
     author: 'Достоевский Ф.М.',
     price:  234.55,
     amount: 8
    }
    ];

let groupObj = d3.group(books, d => d.author);
```

В **groupObj** будет занесена информация следующей структуры (**map**, состоящий из трех элементов, каждый элемент массива – объект, ключ которого – фамилия автора, а значения – информация о книгах, написанных этим автором):

```
0: {"Булгаков М.А." => Array(2)}
      key: "Булгаков М.А."
      value: Array(2)
       0:
            amount: 10
            author: "Булгаков М.А."
            price: 581.5
            title: "Мастер и Маргарита"
       1:
            amount: 10
            author: "Булгаков М.А."
            price: 600
            title: "Белая гвардия"
1: {"Толстой Л.Н." => Array(2)}
      key: "Толстой Л.Н."
      value: Array(2)
       0:
```

```
            amount: 10
            author: "Толстой Л.Н."
            price: 899.99
            title: "Война и мир"
      1:
            amount: 15
            author: "Толстой Л.Н."
            price: 450.1
            title: "Анна Каренина"
2:…
```

Далее перебираем всех авторов и выводим информацию о минимальной цене, максимальной цене и стоимости книг каждого автора.

JavaScript-код:

```
let groupObj = d3.group(books, d => d.author);

for(let item of groupObj) {
    //выделяем все цены книг очередного автора и находим мин и макс
    let minMax = d3.extent(item[1].map(d => d.price));

    //находим общую стоимость книг
    let sum = d3.sum(item[1].map(d => d.price * d.amount));

    console.log(`автор: ${item[0]}
минимальная цена книги: ${minMax[0]},
максимальная цена книги: ${minMax[1]},
общая стоимость книг: ${sum}`);
}
```

Результат:

автор: Булгаков М.А.
минимальная цена книги: 581.5,
максимальная цена книги: 600,
общая стоимость книг: 11815

автор: Толстой Л.Н.
минимальная цена книги: 450.1,
максимальная цена книги: 899.99,
общая стоимость книг: 15751.4

автор: Достоевский Ф.М.
минимальная цена книги: 234.55,
максимальная цена книги: 234.55,
общая стоимость книг: 1876.4

Фильтрация данных

Для фильтрации данных используется метод **`filter()`**, который возвращает элементы, подходящие под заданное условие.

Пример. Выведем информацию о книгах в виде таблицы, а затем реализуем фильтрацию книг по фамилии автора.

HTML-код:

```
<head>
  <script src="http://d3js.org/d3.v7.min.js"> </script>
  <style>
```

```
    table{
      margin: 20px 5px;
      border-collapse:collapse;
    }
    table td {
      padding: 5px 10px;
      border: thin solid grey;
    }
    table tr:nth-child(odd){
       background: LightGrey
    }
</style>
</head>
<body>
  <div class="main">
    <p>Список книг</p>
    <table>
    </table>
  </div>
</body>
  <script src="program.js"> </script>
```

Шаг 1. Выведем всю информацию о книгах в виде таблицы, устанавливаем, что все строки – видимые.

JavaScript-код:

```
let books = [
   …
    ];

// создание таблицы
let table = d3.select("div.main")
    .select("table")

// создание строк таблицы (столько, сколько элементов в массиве books)
let rows = table.selectAll("tr")
    .data(books)
    .enter()
    .append('tr')
    .style("display", "");

// создание ячеек каждой строки на основе каждого элемента массива books
let cells = rows.selectAll("td")
      .data(d => Object.values(d))
      .enter()
      .append("td")
      .text(d => d);
```

Результат:

Список книг:

Мастер и Маргарита	Булгаков М.А.	581.5	10
Белая гвардия	Булгаков М.А.	600	10
Война и мир	Толстой Л.Н.	899.99	10
Анна Каренина	Толстой Л.Н.	450.1	15
Игрок	Достоевский Ф.М.	234.55	8

Для того чтобы вывести **заголовки столбцов таблицы** (ключи элемента массива) используется следующий код:

```
let head = table.insert("tr", "tr")
           .selectAll("th")
           .data(d => Object.keys(books[0]))
           .enter()
           .append("td")
           .text(d => d);
```

Важно! Для упрощения в примере заголовки столбцов ВСТАВЛЯТЬ не будем.

Шаг 2. Выведем книги Толстого Л.Н., для этого отберем все строки, в которых отображаются книги НЕ Толстого и скроем их.

JavaScript-код:

```
d3.select("table")
  .selectAll("tr")
  .filter(d => !(d.author == "Толстой Л.Н."))
  .style("display", "none");
```

Результат:

Список книг:

Война и мир	Толстой Л.Н.	899.99	10
Анна Каренина	Толстой Л.Н.	450.1	15

Шаг 3. Добавим поле со списком, в котором будут отображаться фамилии всех авторов.

HTML-код:

```
<body>
  <div class="main">
    <form>
      <p>Выберите автора:
        <select id="select">
        </select>
      </p>
    </form>
    <p>Список книг:</p>
    <table>
    </table>
  </div>
<body>
```

JavaScript-код:

```
//отбираем всехразличных авторов
let groupObj = d3.group(books, d => d.author)
let author = [...groupObj.keys()];

// создаем поле со списком
let selectAuthor = d3.select("div.main")
    .select("select")

// формируем option
```

```
selectAuthor
    .selectAll("option")
    .data(author)
    .enter()
    .append('option')
    .text( d => d);

// добавляем опцию "Все авторы", отвечаем, что она выбрана
selectAuthor
    .insert("option", "option")
    .attr("selected", "selected")
    .text("Все авторы");
```

Шаг 4. Свяжем вызов функции **`filter()`** с событием ***change*** поля со списком **`selectAuthor`:**

JavaScript-код:

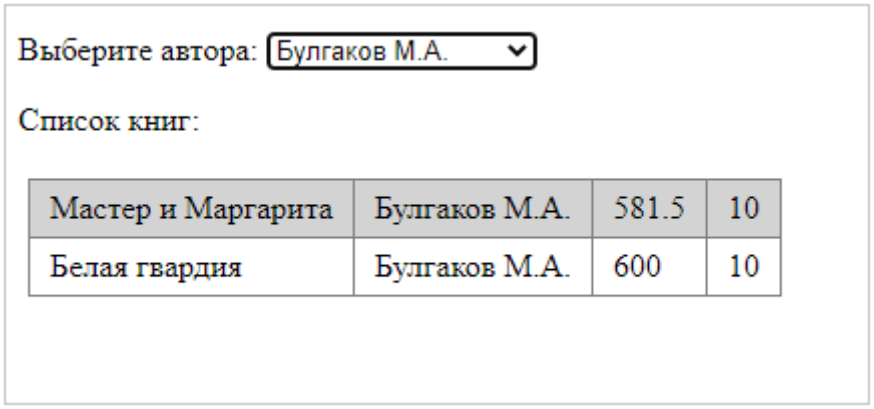
```
selectAuthor
    .on('change', function(){
        let selected =  d3.select(this)
                        .property("value");
        filter(selected)
     });
```

Шаг 5. Реализуем функцию **`filter()`** которая отбирает и выводит на страницу либо все книги, либо книги заданного автора в зависимости от значения параметра.

JavaScript-код:

```
function filter(selectAuthor) {
      // все строки делаем видимыми
      d3.select("table")
        .selectAll("tr")
        .style("display", "");

      //делаем невидимыми все строки, кроме нужных
      d3.select("table")
        .selectAll("tr")
        .filter(d =>
         (selectAuthor !== 'Все авторы') ? !(d.author == selectAuthor) : false
        )
      .style("display", "none");
}
```

Результат:



Сортировка данных

Для сортировки используется метод **sort()**, которая требует функцию для сравнения данных.

Пример. Отсортируем выводимые на страницу книги по их названию в алфавитном порядке.

```
let compareByTitle = (a, b) => {
  return (a.title < b.title) ? -1 : 1
};

d3.select("table")
  .selectAll("tr")
  .sort(compareByTitle)
```

В `compareByTitle` сохраняем функцию, которая сравнивает название книг и возвращает 1 или -1. В данном случае названия сортируются в алфавитном порядке. Если требуется обратный порядок сортировки, то в функции необходимо поменять знак сравнения.

Функция может реализовывать и более сложную сортировку, например, «двухуровневую».

Пример. Отсортируем выводимые на страницу книги сначала по фамилии автора в алфавитном порядке, а затем по убыванию цены.

```
let compareByAuthorPrice = function (a, b) {
  if (a.author < b.author) return -1;
  if (a.author > b.author) return 1;
  return ( a.price >= b.price) ? -1 : 1
};
```

Результат:

Выберите автора: Все авторы

Список книг:

Белая гвардия	Булгаков М.А.	600	10
Мастер и Маргарита	Булгаков М.А.	581.5	10
Игрок	Достоевский Ф.М.	234.55	8
Война и мир	Толстой Л.Н.	899.99	10
Анна Каренина	Толстой Л.Н.	450.1	15

Конструкция scale

Конструкция **scaleLinear** позволяет один интервал значений сопоставить другому интервалу значений, применяя линейное преобразование. Таким образом **scaleLinear** преобразует некоторое значение из одного интервала, который называется **domain**, в значение, принадлежащее другому интервалу, который называется **range**.

Пример. Пусть цена на некоторый товар изменяется с течением времени. Ее значения за полгода следующие 256.12, 278.50, 250.20, 262.00, 275.90, 270.50. Эти значения необходимо отобразить на графике по оси OY. При этом известно, что цены необходимо отобразить на отрезке от 0 до 200.

Решение.

1. Найдем минимальное и максимальное значение цены (250.20 и 278.50)
2. Определим диапазон цены, уменьшим минимальную цену на 2% и увеличим максимальную тоже на 2%, получим диапазон [245, 283]. Этот интервал и есть **domain**.
3. Определим второй интервал, на который нужно отобразить цены. Так как направление оси OY в SVG – сверху вниз, то этот интервал [200, 0]. Этот интервал называется **range**.
4. Для каждого значения цены из **domain** необходимо подобрать значение из **range**, используя линейное сопоставление. Для этого используется **scaleLinear**.

JavaScript-код:

```
let data =[256.12, 278.50, 250.20, 262.00, 275.90, 270.50];

let newData = d3.scaleLinear()
                .domain([245, 283])
                .range([200, 0]);
for(let i in data) {
    document.write(newData(data[i])+', ');
}
```

Результат:

141.47368421052627, 23.684210526315795, 172.63157894736847, 110.52631578947367, 37.368421052631696, 65.7894736842

Числа получаются при преобразовании дробными, округляются при выводе. Если использовать **rangeRound()** вместо **range()**, при преобразовании происходит и округление чисел.

JavaScript-код:

```
let data =[256.12, 278.50, 250.20, 262.00, 275.90, 270.50];

let newData = d3.scaleLinear()
                .domain([245, 283])
                .rangeRound([200, 0]);
for(let i in data) {
    document.write(newData(data[i])+', ');
}
```

Результат:

141, 24, 173, 111, 37, 66,

В примере в качестве способа преобразования использовалась линейная функция. Но можно использовать и ряд других: степень, логарифм и др.

Пример. Построим шкалу для вывода значений о количестве хостов с 1980 по 2005 год. Их значения: 102, 1000, 10000, 6000000, 80000000, 50000000000.

Для отображения подобных значений на графике, использовать линейную шкалу нельзя, в таких случаях используется логарифмическая шкала. Отобразим эти значения на интервал [0, 200].

JavaScript-код:

```
let data =[102, 1000, 10000, 6000000, 80000000, 50000000000];

let newData = d3.scaleLog()
                .domain([50, 60000000000])
                .rangeRound([0, 200]);
for(let i in data) {
    document.write(newData(data[i])+', ');
}
```

Результат:

7, 29, 51, 112, 137, 198,

Функция **d3.scaleTime()** позволяет сопоставлять линейные интервалы с временными отрезками.

Пример. Сопоставим интервал дат значениям по оси OX от 0 до 300.

JavaScript-код:

```
let data = [
    new Date(2023, 2, 11),
    new Date(2023, 2, 26),
    new Date(2023, 3, 1),
    new Date(2023, 3, 8),
    new Date(2023, 3, 28)
];

let start = new Date(2023, 2, 1);
let end = new Date(2023, 4, 1);

let newData = d3.scaleTime()
                .domain([start, end])
                .rangeRound([0, 300]);

for(let i in data) {
    document.write(newData(data[i])+', ');
}
```

Результат:

46, 97, 153, 193, 234

Графические элементы

Для создания графических элементов на html-странице d3 использует элемент SVG, атрибутами которого являются его ширина и высота. Его можно создать либо в программе, либо вставив компонент непосредственно на страницу.

Пример. Создадим SVG –элемент шириной 500 пикселей, высотой 300 пикселей.

JavaScript-код:

```
let width = 500;
let height = 300;

let svg = d3.select("body")
            .append("svg")
            .attr("height", height)
            .attr("width", width);
```

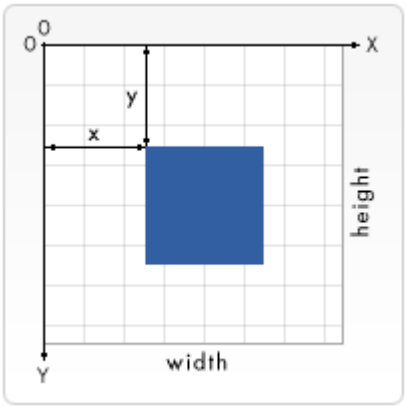
HTML-код:

```
<svg>
</svg>
```

Для созданного элемента можно задать CSS-стиль, например, указать цвет и толщину границы.

```
svg.style("border", "solid grey thin")
```

На SVG-элемент накладывается координатная сетка. Обычно 1 единица на сетке соответствует 1 пикселю. Начало координат сетки расположено в верхнем левом углу SVG-элемента. Все графические элементы располагаются относительно этой точки.



Графические примитивы

Линия

Для создания линий используется элемент **`line`**. Его необходимо добавить в SVG-элемент. При создании линии нужно указать координаты ее концов `(x1, y1)` и `(x2, y2)`.

JavaScript-код:

```
svg.append("line")
   .attr("x1", 20)
   .attr("y1", 50)
   .attr("x2", 300)
   .attr("y2", 200);
```

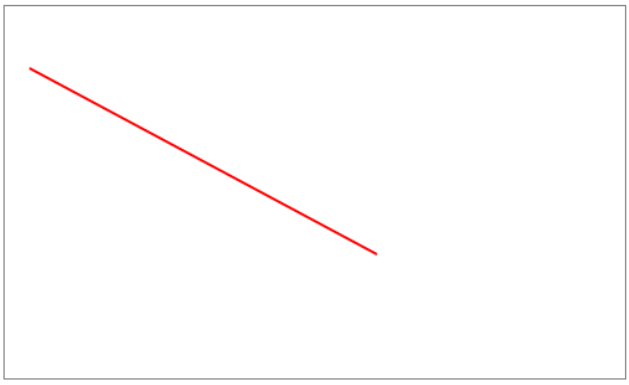
HTML-код:

```
<line x1="20" y1="30" x2="300" y2="300"></line>
```

Для того, чтобы линия отобразилась в элементе, необходимо задать ее стиль: толщину и цвет. Это можно сделать либо через стили CSS, либо в программе:

```
svg.style("stroke", "red")
   .style("stroke-width", "2");
```

Результат в браузере:



Прямоугольник

Для создания прямоугольника применятся элемент **`rect`**. При добавлении прямоугольника надо указать атрибуты **`x`** и **`y`**, указывающие на координаты левого верхнего угла, а также атрибуты **`width`** и **`height`**. При необходимости можно указать радиус скругления углов с помощью атрибута **`rx`**.

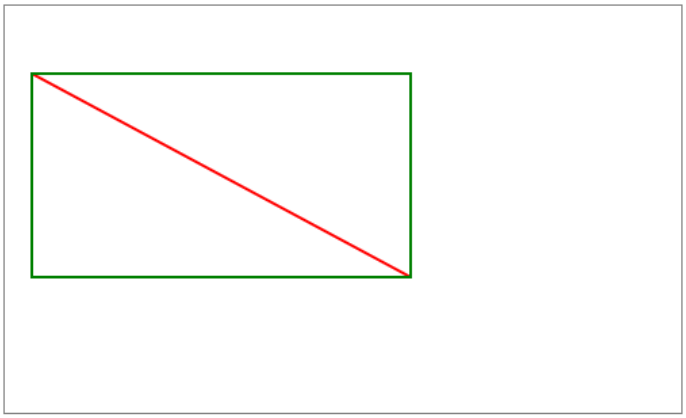
JavaScript-код:

```
svg.append("rect")
    .attr("x", 20)
    .attr("y", 50)
    .attr("width", 280)
    .attr("height", 150)
    .style("fill", "none")
    .style("stroke", "green")
    .style("stroke-width", "2");
```

HTML-код:

```
<line x1="20" y1="30" x2="300" y2="300"></line>
 <rect x="20" y="50" width="280" height="150"
       style="fill: none; stroke: green; stroke-width: 2;">
</rect>
```

Результат в браузере:



Круг

Для создания круга используется элемент **circle**. Для построения круга необходимо указать координаты центра(**cx** и **cy**), а также радиус **r**.

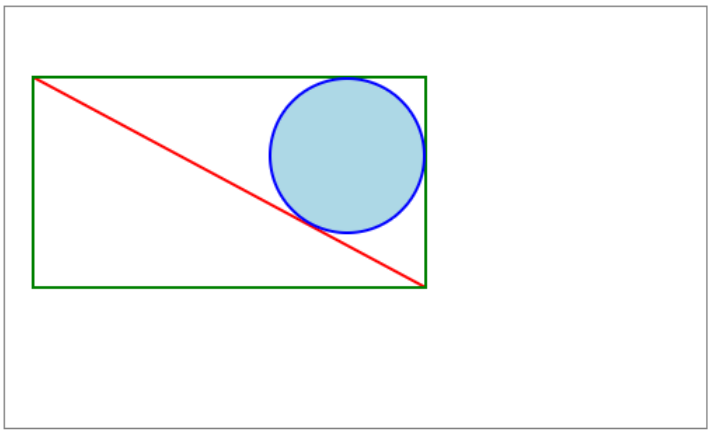
JavaScript-код:

```
svg.append("circle")
    .attr("cx", 244)
    .attr("cy", 106)
    .attr("r", 55)
    .style("fill", "lightblue")
    .style("stroke", "blue")
    .style("stroke-width", "2");
```

HTML-код:

```
<line x1="20" y1="30" x2="300" y2="300"></line>
 <rect x="20" y="50" width="280" height="150"
      style="fill: none; stroke: green; stroke-width: 2;">
</rect>
<circle cx="244" cy="106" r="55"
      style="fill: lightblue; stroke: blue; stroke-width: 2;">
</circle>
```

Результат в браузере:



Многоугольник

Для создания многоугольников используется элемент **polygon**. Атрибут **points** задает массив точек, по которым строится многоугольник. Последняя точка соединяется с первой.

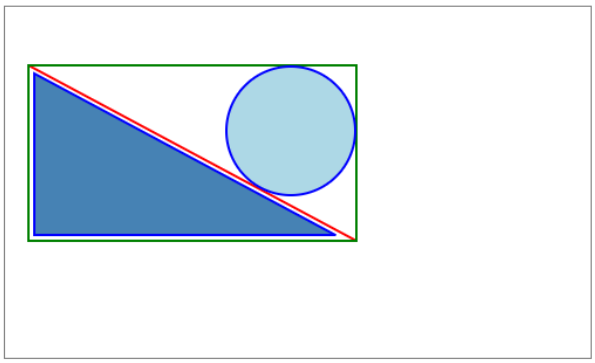
JavaScript-код:

```
 svg.append("polygon")
     .attr("points", "25,57 25,195 282,195")
     .style("fill", "steelblue")
     .style("stroke", "blue")
     .style("stroke-width", "2");
```

HTML-код:

```
…
<polygon points="25,57 25,195 282,195"
      style="fill: steelblue; stroke: blue; stroke-width: 2;">
</polygon>
```

Результат в браузере:



Пути

Путь позволяет отобразить набор любых примитивов с заполнением или без заполнения цветом. Для создания пути используется элемент **path**. Для определения содержимого пути используется атрибут **d**, который кроме точек, может включать следующие команды:

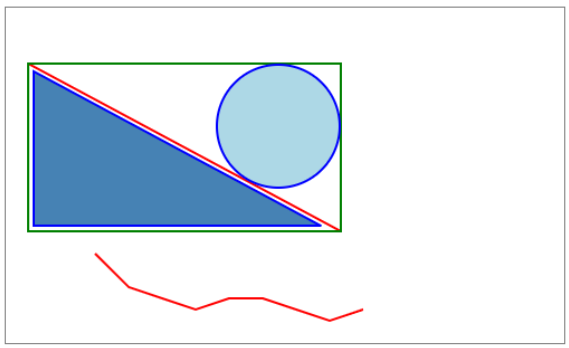
- M(абсолютные координаты) / m(относительные) - переместить в позицию (x y);
- Z / z: завершение пути;
- L / l: провести линию в точку (x y);

```
let data = [
    {x: 80, y: 220},{x: 110, y: 250},{x: 140, y: 260},
    {x: 170, y: 270},{x: 200, y: 260},{x: 230, y: 260},
    {x: 260, y: 270},{x: 290, y: 280},{x: 320, y: 270}
];

// функция, создающая по массиву точек линии
let line = d3.line()
            .x(function(d){return d.x;})
            .y(function(d){return d.y;});

// добавляем путь,
// при использовании линии первая точка записывается с командой M
// остальные - с командой L
svg.append("path")
   .attr("d", line(data))
   .style("fill", "none")
   .style("stroke", "red")
   .style("stroke-width", "2")
```

HTML-код:

```
…
<path
  d="M80,220L110,250L140,260L170,270L200,260L230,260L260,270L290,280L320,270"
  style="fill: none; stroke: red; stroke-width: 2;">
</path>
```

Результат в браузере:



Группы объектов

D3 предоставляет возможность создания групп объектов. Такие группы задаются с помощью элемента **g**. В него можно добавлять примитивы и пути.

JavaScript-код:

```
let group = svg.append("g");
group.append("line")
     .attr("x1", 20)
     .attr("y1", 50)
     .attr("x2", 300)
     .attr("y2", 200)
     .style("stroke", "red")
     .style("stroke-width", "2");

group.append("rect")
     .attr("x", 20)
     .attr("y", 50)
     .attr("width", 280)
     .attr("height", 150)
     .style("fill", "none")
     .style("stroke", "green")
     .style("stroke-width", "2");

group.append("circle")
     .attr("cx", 244)
     .attr("cy", 106)
     .attr("r", 55)
     .style("fill", "lightblue")
     .style("stroke", "blue")
     .style("stroke-width", "2");

group.append("polygon")
     .attr("points", "25,57 25,195 282,195")
     .style("fill", "steelblue")
     .style("stroke", "blue")
     .style("stroke-width", "2")
```

Результат получится такой же, как при прямом добавлении примитивов в элемент SVG.

Групп в svg, задаваемых элементом **g**, может быть несколько. Преимущество использования групп заключается в том, что, с одной стороны, можно применить трансформации к группе элементов, а не к каждому примитива. А с другой эти трансформации не будут применяться к другим группам.

Трансформации и перемещения

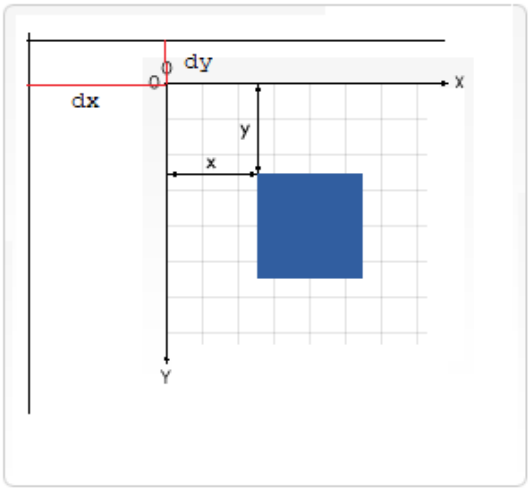
В D3.js определены три типа трансформаций:

- перемещение;
- масштабирование;
- вращение.

Все трансформации задаются для элемента с помощью атрибута **`transform`**.

Перемещение

Трансформацию перемещение можно рассматривать как относительную систему координат: начало координат смешается в точку (**`dx, dy`**) и все координаты точек вычисляются относительно этой системы координат.



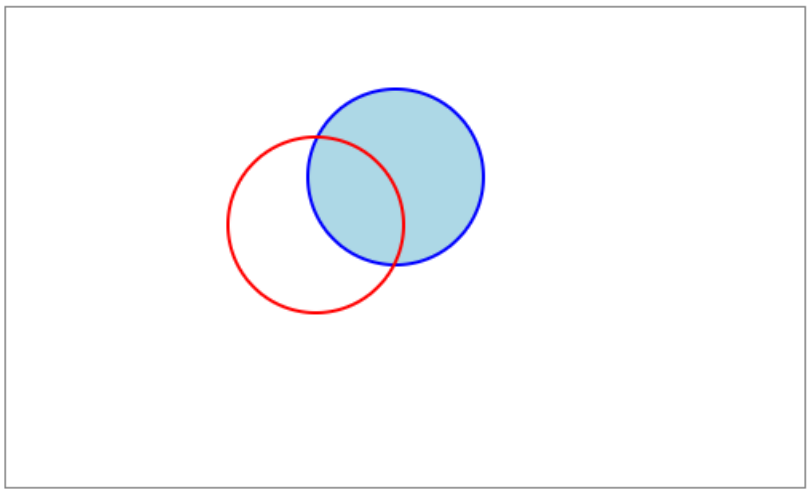
Так реальное значение координат в SVG-элементе верхнего левого угла прямоугольника **(x+dx, y+dy)**. Но в относительной системе координат, полученной при трансформации перемещения, координаты **(x, y)**.

Для перемещения элементов или групп элементов атрибут **transform** принимает значение **translate(dx, dy)**.

JavaScript-код:

```
//круг в системе координат SVG-элемента
svg.append("circle")
    .attr("cx", 244)
    .attr("cy", 106)
    .attr("r", 55)
    .style("fill", "lightblue")
    .style("stroke", "blue")
    .style("stroke-width", "2");

//круг с перемещением
svg.append("circle")
    .attr("cx", 244)
    .attr("cy", 106)
    .attr("r", 55)
    .attr("transform", "translate(-50, 30)")
    .style("fill", "none")
    .style("stroke", "red")
    .style("stroke-width", "2");
```

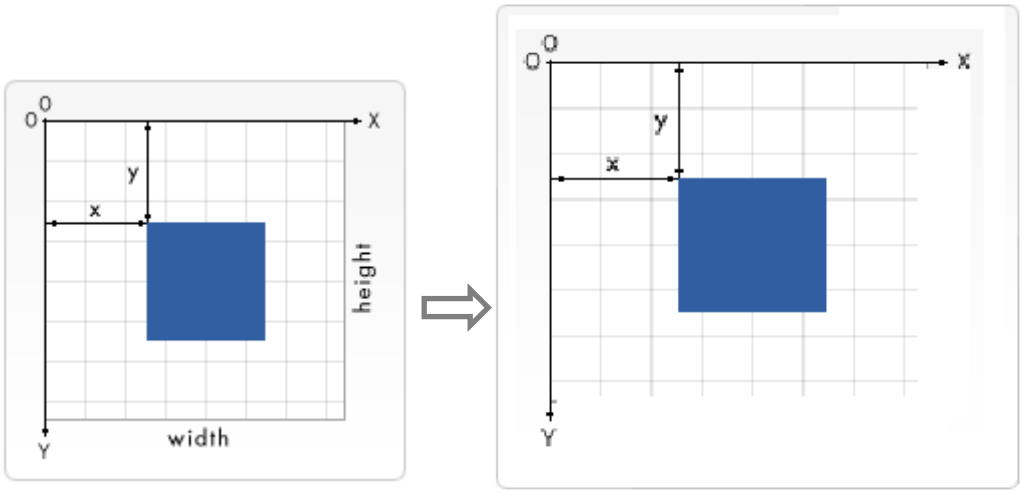
Результат:



Координаты обоих фигур одинаковы, но они отображаются в различных системах координат. При перемещении система координат смещена относительно основной влево на 50 пикселей и вниз на 30.

Масштабирование

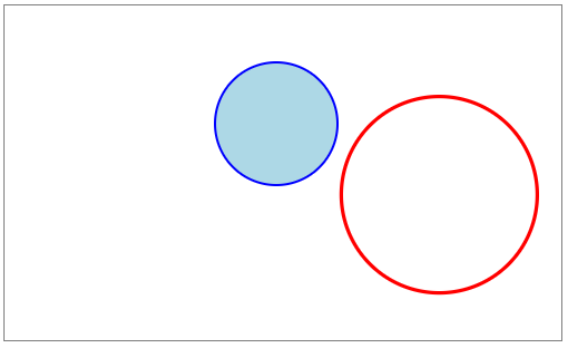
Для масштабирования используется значение атрибута **`transform scale(s)`** (увеличение в **`s`** раз и по горизонтали, и по вертикали) или **`scale(sx, sy)`** (увеличение по горизонтали в **`sx`** раз и по вертикали в **`sy`** раз). Трансформация масштабирования тоже применяется к системе координат, то есть единичный отрезок в системе увеличивается в заданное число раз:



JavaScript-код:

```
//круг в системе координат SVG-элемента
svg.append("circle")
    .attr("cx", 244)
    .attr("cy", 106)
    .attr("r", 55)
    .style("fill", "lightblue")
    .style("stroke", "blue")
    .style("stroke-width", "2");
//круг в системе координат с увеличебнием в 1.6 раз
svg.append("circle")
    .attr("cx", 244)
    .attr("cy", 106)
    .attr("r", 55)
    .attr("transform", "scale(1.6)")
    .style("fill", "none")
    .style("stroke", "red")
    .style("stroke-width", "2");
```

Результат:



В 1.6 раз увеличился не только радиус, но значение координат центра тоже увеличились в 1.6 раз.

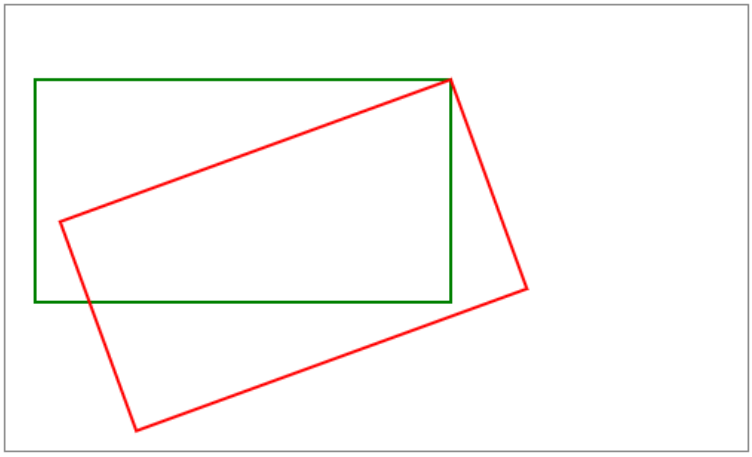
Вращение

Для вращения фигур применяется значение **`rotate(degree,x,y)`**, где первый параметр - угол в градусах поворота по часовой стрелке, а **`x`** и **`y`** - координаты центра точки поворота.

JavaScript-код:

```
svg.append("rect")
     .attr("x", 20)
     .attr("y", 50)
     .attr("width", 280)
     .attr("height", 150)
     .style("fill", "none")
     .style("stroke", "green")
     .style("stroke-width", "2");

// прямоугольник с поворотом относительно
// верхнего правого угла на 20 градусов
// против часовой стрелки
svg.append("rect")
     .attr("x", 20)
     .attr("y", 50)
     .attr("width", 280)
     .attr("height", 150)
     .attr("transform", "rotate(-20, 300, 50)")
     .style("fill", "none")
     .style("stroke", "red")
     .style("stroke-width", "2");
```

Результат:



Применение нескольких трансформаций одновременно

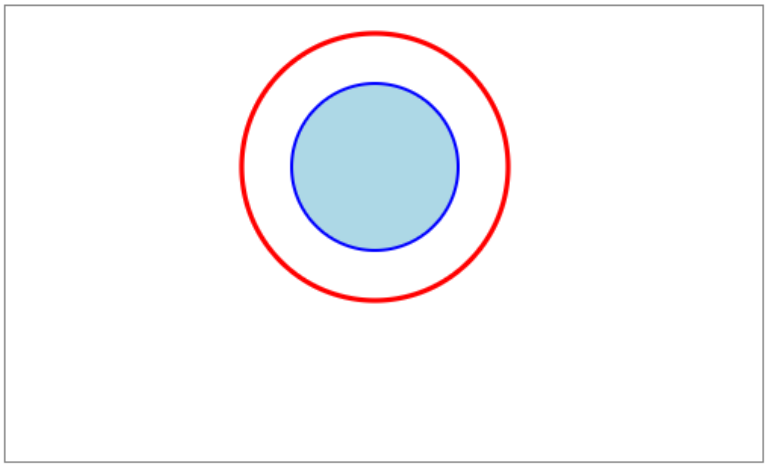
При необходимости можно применять все или несколько трансформаций сразу.

JavaScript-код:

```
//круг в системе координат SVG-элемента
svg.append("circle")
     .attr("cx", 244)
     .attr("cy", 106)
     .attr("r", 55)
     .style("fill", "lightblue")
```

```
    .style("stroke", "blue")
    .style("stroke-width", "2");

//круг с увеличебнием в 1.6 раз
// относительно его центра
svg.append("circle")
    .attr("cx", 0)
    .attr("cy", 0)
    .attr("r", 55)
    .attr("transform", "translate(244, 106) scale(1.6)")
    .style("fill", "none")
    .style("stroke", "red")
    .style("stroke-width", "2");
```

Результат:



Переходы

Переходы (**transitions**) в D3.js реализует динамическое изменение свойств объекта в течение некоторого промежутка времени, то есть его анимацию. Для создания перехода используются следующие методы:

- **transition()** – указывает необходимость перехода элемента из одного состояния в другое
- **duration(duration)** - устанавливает время перехода;
- **delay(duration)** - устанавливает период задержки перед выполнением перехода, может отсутствовать.

Параметр **duration** задает время в миллисекундах

Суть перехода заключается в следующем:

- создается элемент, описываются его атрибуты и стили в начальном состоянии;
- к элементу применяется методы **transition()** и **duration()**, возможно **delay()**.
- для элемента указываются атрибуты и стили, которые должны быть у него в конечном состоянии.

Библиотека автоматически рассчитает промежуточные показатели, которые будут принимать стили или атрибуты, имеющие разные значения в начальном и конечном состоянии.

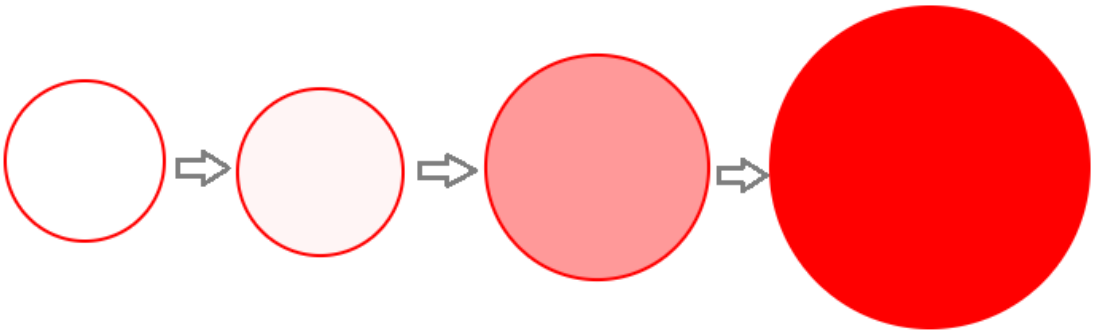
В следующем примере d3 рассчитывает значения, которые будет принимать стилевое свойство **fill** между значениями белым и красным, а также радиус круга от 50 до 100 пикселей на протяжении 4000 милисекунд.

JavaScript-код:

```
svg.append("circle")
```

```
    .attr("cx",250)
    .attr("cy", 150)
    .attr("r", 50)
    .style("fill", "white")
    .style("stroke", "red")
    .style("stroke-width", "2")
    .transition()
    .duration(4000)
    .attr("r", 100)
    .style("fill", "red");
```

Результат:



Графики и диаграммы

Графики и диаграммы строятся по некоторому массиву данных. В данном разделе в качестве примера будем строить график функции на заданном интервале **[a, b].**

$$f(x) = x^3 - 6x^2 + x + 5$$

JavaScript-код:

```
function f(x) {
    return x ** 3 - 6 * x ** 2 + x + 5;
}

a = -2;
b = 6;
n = 30;
h = (b - a) /(n - 1);

let arrGraph =[];
for(let i = 0; i < n; i++) {
    let x =  a + i * h;
    arrGraph.push({'x': x, 'f': f(x)});
}
```

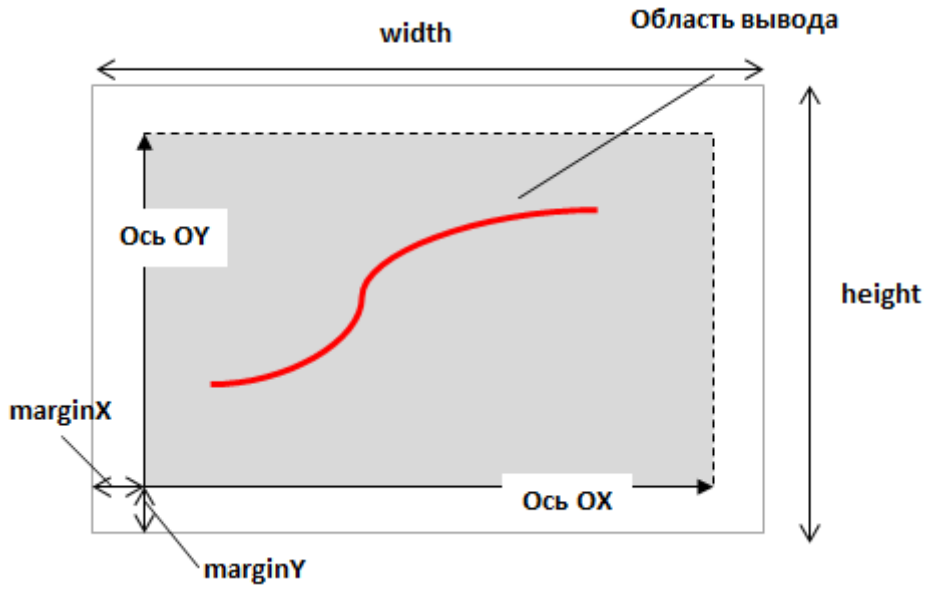
Графики функций расположим в SVG-элементе:

JavaScript-код:

```
let width = 500;
let height = 500;

let svg = d3.select("body")
            .append("svg")
            .attr("height", height)
            .attr("width", width)
            .style("border", "solid thin grey");
```

Для вывода графика в SVG-элемент необходима область построения, которая включает:

- оси координат;
- линии графиков;
- подписи осей координат;
- легенда
- и пр



Элементы графика будем размещать в области, закрашенной серым цветом. Отступы от границ SVG-элемента определим в переменных **marginX, marginY**:

JavaScript-код:

```
let marginX = 50;
let marginY = 40;
```

Оси координат

Для построения и вывода осей координат в библиотеке d3.js используется метод **d3.svg.axis()**.

Для создания осей координат необходимо:

1. Определить интервал значений данных, которые будут отображать на графике. В нашем случае:
   - по оси OX: интервал **[a, b]**;
   - по оси OY: интервал от минимального значения функции до максимального.
2. Определить размеры области для отображения осей на графике:
   - по оси OX: **[0, width – 2 * marginX]**;
   - по оси OY: **[height – 2 * marginY, 0].**
3. Сопоставить значения функции с помощью линейной интерполяции.
4. Создать оси, объединив элементы каждой в группу.
5. Поместить оси в SVG-элементе, сдвинув их в нужное место области построения:
   - по оси OX: смещение (**marginX**, **height – marginY**);
   - по оси OY: смещение (**marginX**, **marginY**).

JavaScript-код:

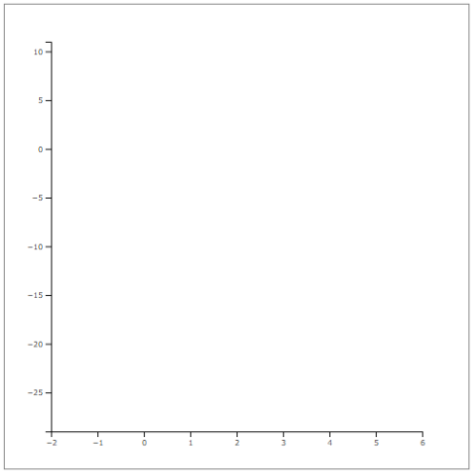
```
let minMaxF = d3.extent(arrGraph.map(d => d.f));
let min = minMaxF[0];
let max = minMaxF[1];

// функция интерполяции значений на оси
let scaleX = d3.scaleLinear()
 .domain([a, b])
 .range([0, width - 2 * marginX]);

let scaleY = d3.scaleLinear()
 .domain([min, max])
 .range([height - 2 * marginY, 0]);

// создание осей
let axisX = d3.axisBottom(scaleX); // горизонтальная

let axisY = d3.axisLeft(scaleY); // вертикальная

// отрисовка осей в SVG-элементе
svg.append("g")
 .attr("transform", `translate(${marginX}, ${height - marginY})`)
 .call(axisX);

svg.append("g")
 .attr("transform", `translate(${marginX}, ${marginY})`)
 .call(axisY);
```

Чтобы не задавать стили каждой оси, добавим CSS-стиль в HTML-документ:

```
<style>
```

```
  svg path, line {
    fill: none;
    stroke: black;
    width: 1px;
  }

  svg text {
    font: 8px Verdana;
}
</style>
```

Результат:

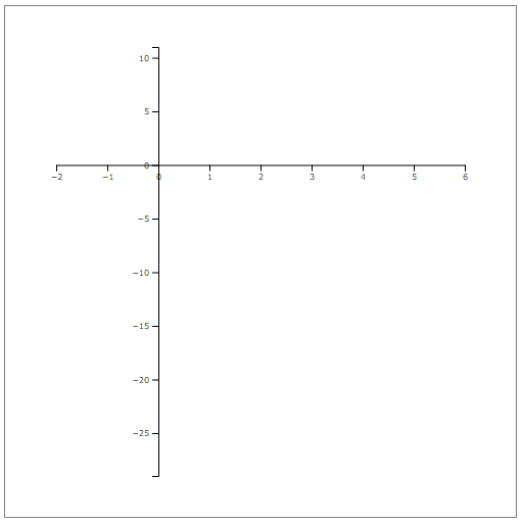


Для вывода графиков, пересечение осей координат привычнее располагать в точке (0, 0). Для перемещения осей в эту точку, нужно определить ее положение в области построения графиков с помощью функций интерполяции:

JavaScript-код:

```
svg.append("g")
    .attr("transform", `translate(${marginX}, ${scaleY(0) + marginY})`)
    .call(axisX);
svg.append("g")
    .attr("transform", `translate(${marginX + scaleX(0)}, ${marginY})`)
    .call(axisY);
```

Результат:



Построение графика

Для создания линии графика **f(x)** необходимо:

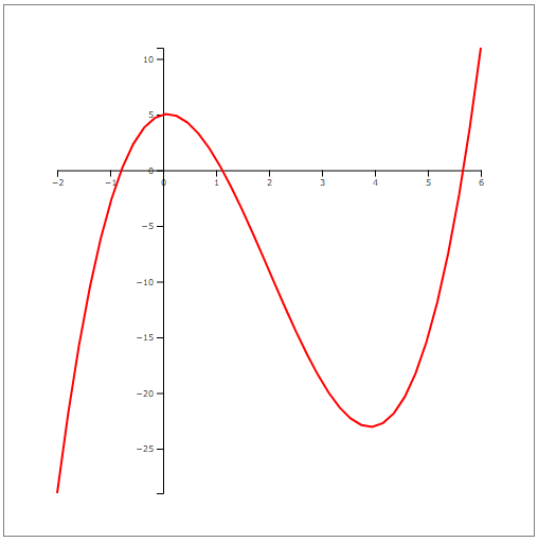
1. Создать путь из линий, отображающий набор точек из массива со значениями функции на графике с помощью функций интерполяции:

```
let lineF = d3.line()
        .x(d => scaleX(d.x))
        .y(d => scaleY(d.f))
```

2. Сформировать и отрисовать линию графика в области построения:

```
svg.append("path") // добавляем путь
// созданному пути добавляются данные массива arrGraph в качестве атрибута
   .datum(arrGraph)
   // вычисляем координаты концов линий с помощью функции lineF
   .attr("d", lineF)
   // помемещаем путь из линий в область построения
   .attr("transform", `translate(${marginX}, ${marginY})`)
    // задаем стиль линии графика
   .style("stroke-width", "2")
   .style("stroke", "red")
```

Результат:



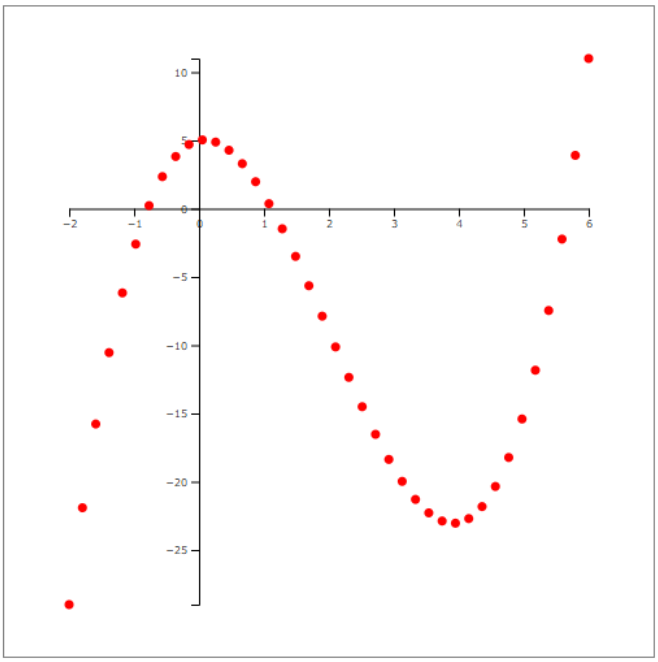
Построение точечной диаграммы

Точечная диаграмма представляет собой совокупность элементов **circle**, выведенных в точках отображения графика в области построения. Координаты каждой точки вычисляются с помощью функций интерполяции.

JavaScript-код:

```
svg.selectAll(".dot")
    .data(arrGraph)
    .enter()
    .append("circle")
    .attr("r", 3.5)
    .attr("cx", d => scaleX(d.x))
    .attr("cy", d => scaleY(d.f))
    .attr("transform", `translate(${marginX}, ${marginY})`)
    .style("fill", "red")
```

Результат:



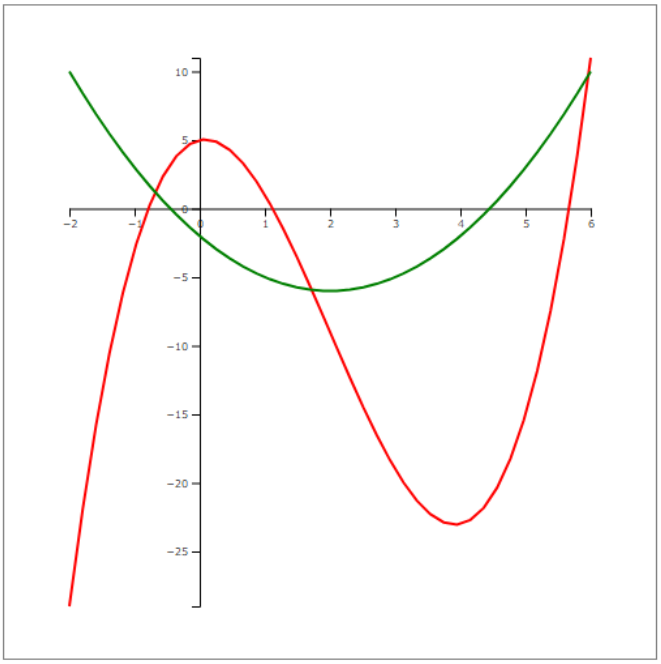
Легенда графиков

На одной области построения можно вывести несколько графиков. Например, отобразим в области построения графики двух функций на одном интервале:

$$f(x) = x^3 - 6x^2 + x + 5$$
$$y(x) = (x-2)^2 - 6$$

График второй функции строится аналогично, только необходимо найти новые минимальные и максимальные значения отображения графика по оси OY с учетом новой функции.

Результат:



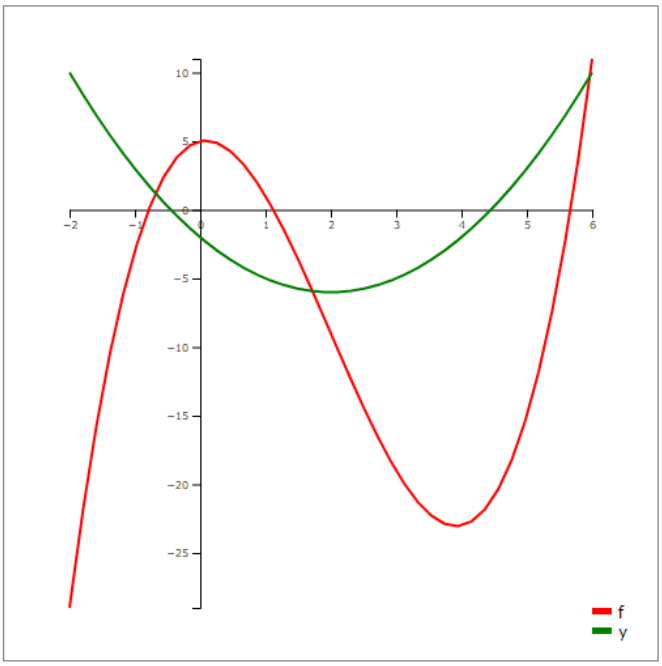
Если в области построения отображается несколько линий, то рекомендуется выводить описание каждой линии – ее легенду. Легенда соотносит используемые на диаграмме цвета с текстовыми метками каждой линии.

Создание легенды представляет собой генерацию прямоугольников, окрашенных в цвета соответствующих линий, и текстовых меток к ним.

JavaScript-код

```
// подписи к графикам
 let legend_keys = ["f", "y"]
// цвета графиков
let color =['red', 'green']

// создаем легенды для каждого графика
var lineLegend = svg
     .selectAll(".lineLegend")
     .data(legend_keys)
     .enter()
     .append("g")
     .attr("class","lineLegend")
     // координаты вывода выбираем произвольно
     .attr("transform", function (d,i) {
        return `translate(${width - marginX}, ${height - marginY - i * 15})`;
    });

// добавляем текстовые надписи
lineLegend.append("text")
          .text(d => d)
          .attr("transform", "translate(20,8)")
          .style("font", "12px Verdana");

// добавляем прямоугольники соответствующих цветов
lineLegend.append("rect")
     .attr("fill", (d, i) => color[i])
     .attr("width", 15).attr("height", 5);
```

Результат:



Построение гистограммы

Построение гистограммы выполним для следующих данных:

```
let authors = [
    {
     author: 'Булгаков М.А.',
     amount: 12
    },
    {
     author: 'Толстой Л.Н.',
     amount: 16
    },
    {
     author: 'Достоевский Ф.М.',
     amount: 18
    },
    {
     author: 'Пушкин А.С. ',
     amount: 13
    }
];
```

Для отображения на оси текстовых значений вместо чисел используется метод **d3.scaleBand()**,который сопоставляет размер оси **range()** в пикселях с текстовыми значениями описанными в **domain()**. При этом можно указать отступы между элементами диаграммы.

```
d3.scaleBand()
   .domain(объект.map(d => d.поле))
   .range([0, длинаОсиX])
   .padding(число);
```

Цвета столбиков можно задать либо в виде массива, либо использовать специальную функцию, которая возвращает один из 10 цветов по порядку:

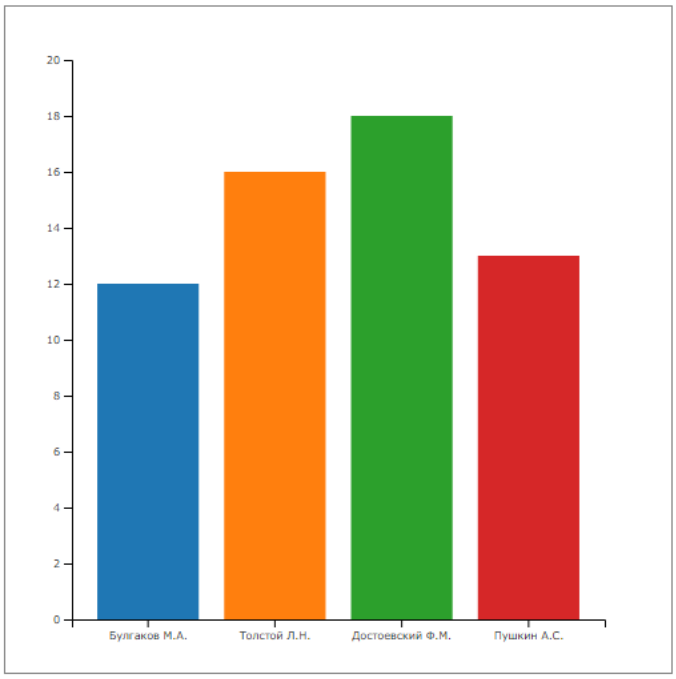
```
d3.scaleOrdinal(d3.schemeCategory10);
```

JavaScript-код (построения гистограммы):

```
let width = 500;
let height = 500;
let marginX = 50;
let marginY = 40;

let svg = d3.select("body")
            .append("svg")
            .attr("height", height)
            .attr("width", width)
            .style("border", "solid thin grey");

let min = 0;
let max = 20;

let xAxisLen = width - 2 * marginX;
let yAxisLen = height - 2 * marginY;

// функции шкалирования
let scaleX = d3.scaleBand()
               .domain(authors.map(d => d.author))
               .range([0, xAxisLen])
```

```
                .padding(0.2);

let scaleY = d3.scaleLinear()
                .domain([min, max])
                .range([yAxisLen, 0]);

 // создание осей
let axisX = d3.axisBottom(scaleX); // горизонтальная

let axisY = d3.axisLeft(scaleY);// вертикальная


svg.append("g")
   .attr("transform", `translate(${marginX}, ${height -  marginY})`)
   .call(axisX)
   .attr("class", "x-axis");

svg.append("g")
   .attr("transform", `translate(${marginX}, ${marginY})`)
   .call(axisY);

//цвета столбиков
let color = d3.scaleOrdinal(d3.schemeCategory10);

//создание и отрисовка столбиков гистограммы
g =svg.append("g")
    .attr("transform",  `translate(${ marginX}, ${ marginY})`)
    .selectAll(".rect")
    .data(authors)
    .enter().append("rect")
    .attr("x", d => scaleX(d.author))
    .attr("width", scaleX.bandwidth())
    .attr("y", d => scaleY(d.amount))
    .attr("height", d => yAxisLen - scaleY(d.amount))
    .attr("fill", d => color(d.author));
```

Результат:



Подписи осей координат и графиков

Добавим подпись оси OY, расположенную слева от оси, снизу вниз.

JavaScript-код (построения гистограммы, продолжение):

```
svg.append("text")
     .attr("x", marginX / 2)
     .attr("y", height / 2 )
     .style("text-anchor", "middle")
     .style("font-size", "12px")
     .attr("transform", `rotate(-90, ${marginX / 2}, ${height / 2})`)
     .text("Количество книг");
```

Результат: