ПМ3 Разработка модулей ПО.

РО 3.2 Разрабатывать модули с применением DOM API, Regexp, HTTP

Тема 1. JS: DOM API.

Лекция 6. Поиск элементов и работа с коллекциями.

Цель занятия:

Научиться находить элементы в DOM разными методами, разбираться в типах коллекций (HTMLCollection, NodeList) и уметь использовать CSS-селекторы и атрибуты для выборки элементов.

Учебные вопросы:

1. Поиск по CSS-селекторам и атрибутам.
2. Практика.

1. Поиск по CSS-селекторам и атрибутам.

Зачем использовать селекторы?

Методы querySelector и querySelectorAll позволяют находить элементы не только по id или классу, а любым CSS-селектором. Это удобно, если:

- элементы сложно отличить только по тегу или классу;
- нужен первый/последний/конкретный потомок;
- требуется фильтрация по атрибутам (href, data-* и др.).

◇ Простые селекторы.

По тегу:

```
document.querySelector("p"); // первый <p>
document.querySelectorAll("p"); // все <p>
```

По классу:

```
document.querySelector(".note"); // элемент с классом note
document.querySelectorAll(".note"); // все элементы с этим классом
```

По id:

```
document.querySelector("#header"); // элемент с id="header"
```

◇ Комбинированные селекторы.

Потомки через пробел:

```
document.querySelector("ul li"); // первый <li> внутри <ul>
```

Непосредственный потомок (>):

```
document.querySelector("div > p"); // <p>, у которого родитель — div
```

Класс + тег:

```
document.querySelector("p.active"); // <p class="active">
```

Псевдоклассы (работают в JS так же, как в CSS):

```
document.querySelector("ul li:first-child"); // первый <li> в списке
document.querySelector("ul li:last-child");  // последний <li>
```

◇ Работа с data-* атрибутами.

HTML5 позволяет создавать пользовательские атрибуты data-*:

```
<div data-role="admin" data-level="5">Пользователь</div>
```

Поиск через селекторы:

```
document.querySelector('[data-role="admin"]'); // элемент с data-role="admin"
```

Доступ к значению в JS:

```
let el = document.querySelector('[data-role="admin"]');
console.log(el.dataset.role);  // "admin"
console.log(el.dataset.level); // "5"
```

Подробнее про **data-*** атрибуты и **dataset**.

🔷 Что такое data-* атрибуты?

- Это специальные пользовательские атрибуты, которые начинаются с префикса data-.
- Стандарт HTML5 разрешает их использовать для хранения «мини-данных» прямо в HTML.
- Они не влияют на отображение страницы, но доступны в JS и CSS.

👉 Используются, когда нужно хранить небольшую информацию, связанную с элементом (например, id товара, роль пользователя, состояние кнопки).

Пример:

```
<button data-id="42" data-role="delete">Удалить</button>
```

Здесь:

data-id="42" — идентификатор (например, товара).

data-role="delete" — назначение кнопки.

◇ Как работать с ними в JS?

1. Через getAttribute:

```
let btn = document.querySelector("button");
console.log(btn.getAttribute("data-id"));   // "42"
console.log(btn.getAttribute("data-role")); // "delete"
```

2. Через свойство dataset.

У каждого элемента есть свойство dataset.

Это объект, где ключи соответствуют data-* атрибутам.

```
let btn = document.querySelector("button");
console.log(btn.dataset.id);   // "42"
console.log(btn.dataset.role); // "delete"
```

⚠ **Внимание:**

В dataset имена пишутся в camelCase:

- data-user-name → dataset.userName
- data-max-value → dataset.maxValue

◇ Использование в CSS.

Атрибуты data-* можно использовать в селекторах:

```
<style>
  button[data-role="delete"] {
    color: red;
  }
</style>
```

```
<button data-role="delete">delete</button>
```

🔷 Когда использовать data-*?

✅ Удобны для «мини-данных» в HTML (id, роль, состояние).

✅ Не требуют дополнительных запросов на сервер.

❌ Но не стоит хранить в них большие объёмы данных или чувствительную информацию (пароли, токены).

👉 Вывод:

- data-* — способ хранить пользовательские данные в HTML.
- В JS доступны через element.dataset как свойства объекта.
- В CSS — через селекторы по атрибутам.

👉 Вывод:

- querySelector* работает со всеми CSS-селекторами.
- Можно комбинировать селекторы для точного поиска.
- data-* атрибуты — гибкий способ хранить «мини-данные» внутри разметки, легко читаются через dataset.

2. Практика работы с DOM: поиск и коллекции.

- 1. Изменить текст элемента по id

```
<h1 id="title">Старый заголовок</h1>
<script>
  let h1 = document.getElementById("title");
  h1.textContent = "Новый заголовок!";
</script>
```

◇ 2. Подсветить группу элементов по классу.

```
<ul>
  <li class="item">Элемент 1</li>
  <li class="item">Элемент 2</li>
  <li class="item">Элемент 3</li>
</ul>

<script>
  let items = document.getElementsByClassName("item");
  for (let el of items) {
    el.style.background = "yellow";
  }
</script>
```

◇ 3. Выделить элементы с data-role и изменить стиль.

```
<button data-role="admin">Admin</button>
<button data-role="user">User</button>

<script>
  let buttons = document.querySelectorAll("[data-role]");
  buttons.forEach(btn => {
    btn.style.border = "2px solid red";
  });
</script>
```

◇ 4. Пройтись циклом по коллекции и добавить изменения.

```
<ul>
  <li>Первый</li>
  <li>Второй</li>
  <li>Третий</li>
</ul>

<script>
  let lis = document.querySelectorAll("li");
  lis.forEach((li, index) => {
    li.textContent = `${index + 1}. ${li.textContent}`;
  });
</script>
```

◇ 5. Комбинированный пример: работа с разными селекторами.

```
<div id="box" class="container" data-type="info">
  Привет, я контейнер!
</div>

<script>
  let box = document.querySelector("#box.container[data-type='info']");
  box.style.color = "blue";
  box.style.fontWeight = "bold";
</script>
```

👉 **Вывод**:

- Методы поиска позволяют находить элементы по id, классу, тегу, селекторам и атрибутам.
- С коллекциями (HTMLCollection, NodeList) можно работать циклом или через forEach.
- Часто практическая задача: найти группу элементов и массово изменить их стиль или текст.

Контрольные вопросы:

- Как выбрать все <p> с классом .active?
- Как обратиться к data-* атрибуту через JS?

Домашнее задание:

1. https://ru.hexlet.io/courses/js-dom