Document Object Model (DOM)

JS COURSE
ORT DNIPRO

ORTDNIPRO.ORG/JS

1. HTML Document

Структура HTML-документа

состоит из:

<tag attr="value”>Text data</tag>

Теги как контейнер для информации
+**атрибуты**

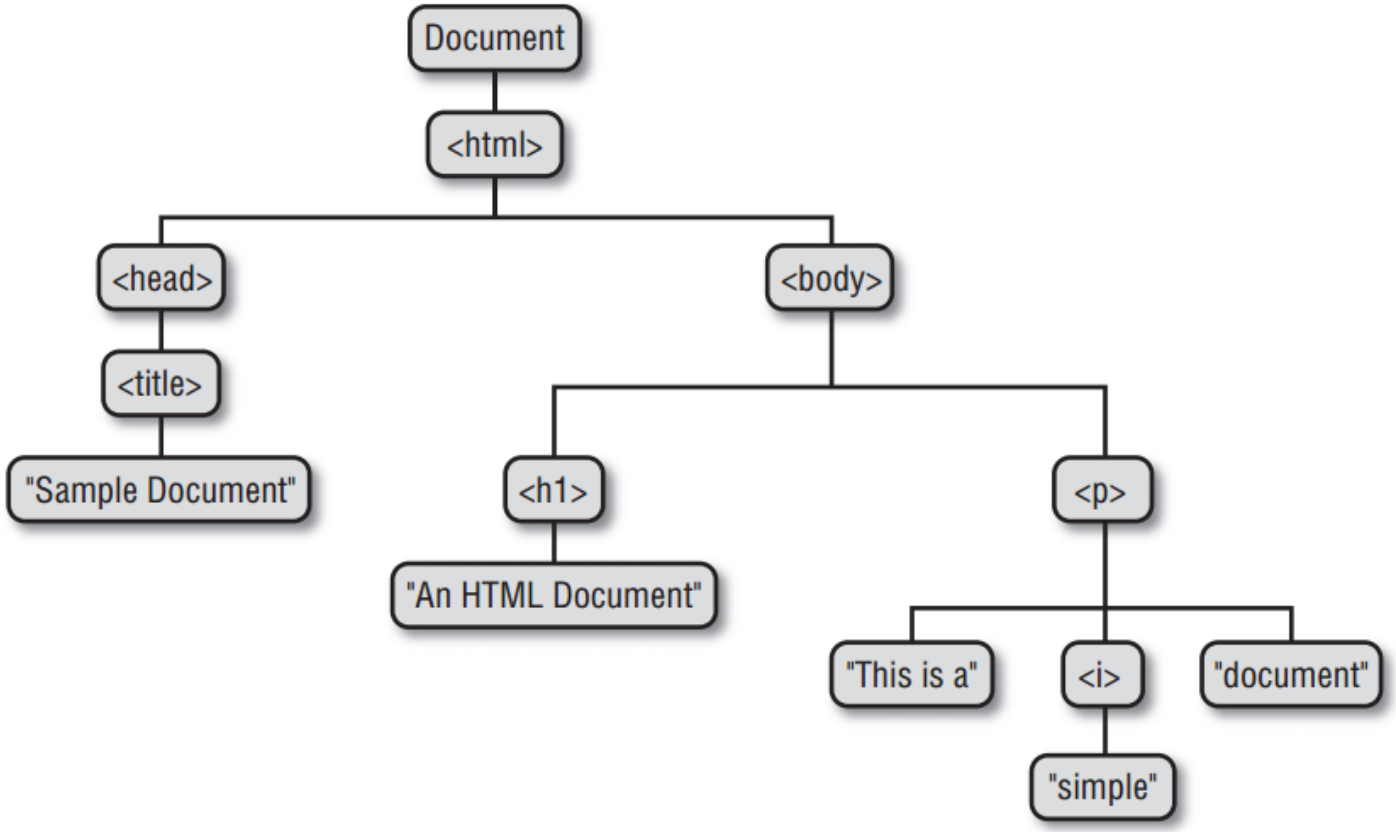
Текстовые данные (содержимое, контент)

Структура HTML-документа

```
1   <!DOCTYPE html>
2   <html lang="en">
3       <head>
4           <title>Sample Document</title>
5       </head>
6       <body>
7           <h1>An HTML Document</h1>
8           <p>This is <i>simple</i> document</p>
9       </body>
10  </html>
```

Древовидная структура HTML-документа

Древовидная структура HTML-документа



В контексте **JavaScript,** каждый **тег** дерева представлен **объектом** (часто используется термин: **узел**, **node**). У каждого элемента есть один **родительский элемент**, и множество **дочерних элементов** (от 0 до ∞).

1. DOM

Document Object Model (DOM)

Объектная Модель Документа

Стандарт определяющий из каких объектов браузер собирает дерево документа, и какие свойства и методы есть у этих объектов.

https://learn.javascript.ru/document

Задача JavaScript – манипуляция HTML-документом

1. *Добавление нового элемента:*
   Создать новый элемент и присоединить его, в качестве дочернего, к одному из существующих элементов;

2. *Изменение элемента:*
   Изменение свойств элемента (в т.ч. содержимого);
   Изменение его позиции в дереве документа;

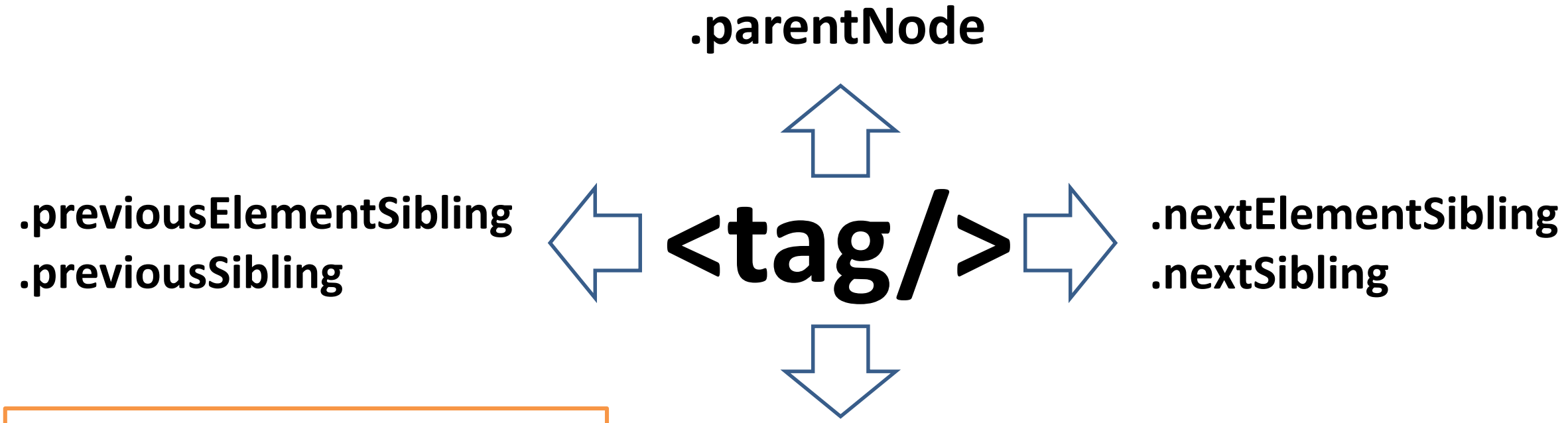
3. *Удаление элемента (из дерева документа).*

Node/Узел/Тег/Элемент

Каждый тег представлен объектом

Воздействие на свойства и методы которого позволяют управлять отображением тега на странице.

Свойства элементов HTML-документа



Каждый **объект** (**элемент**, **тег**) имеет среди своих свойств те которые хранят ссылку на родительский элемент (**parentNode**), на соседние элементы (**previousElementSibling** и **nextElementSibling**) и на перечень потомков (**childNodes** и **children**)

Свойства элементов HTML-документа

<tag/>

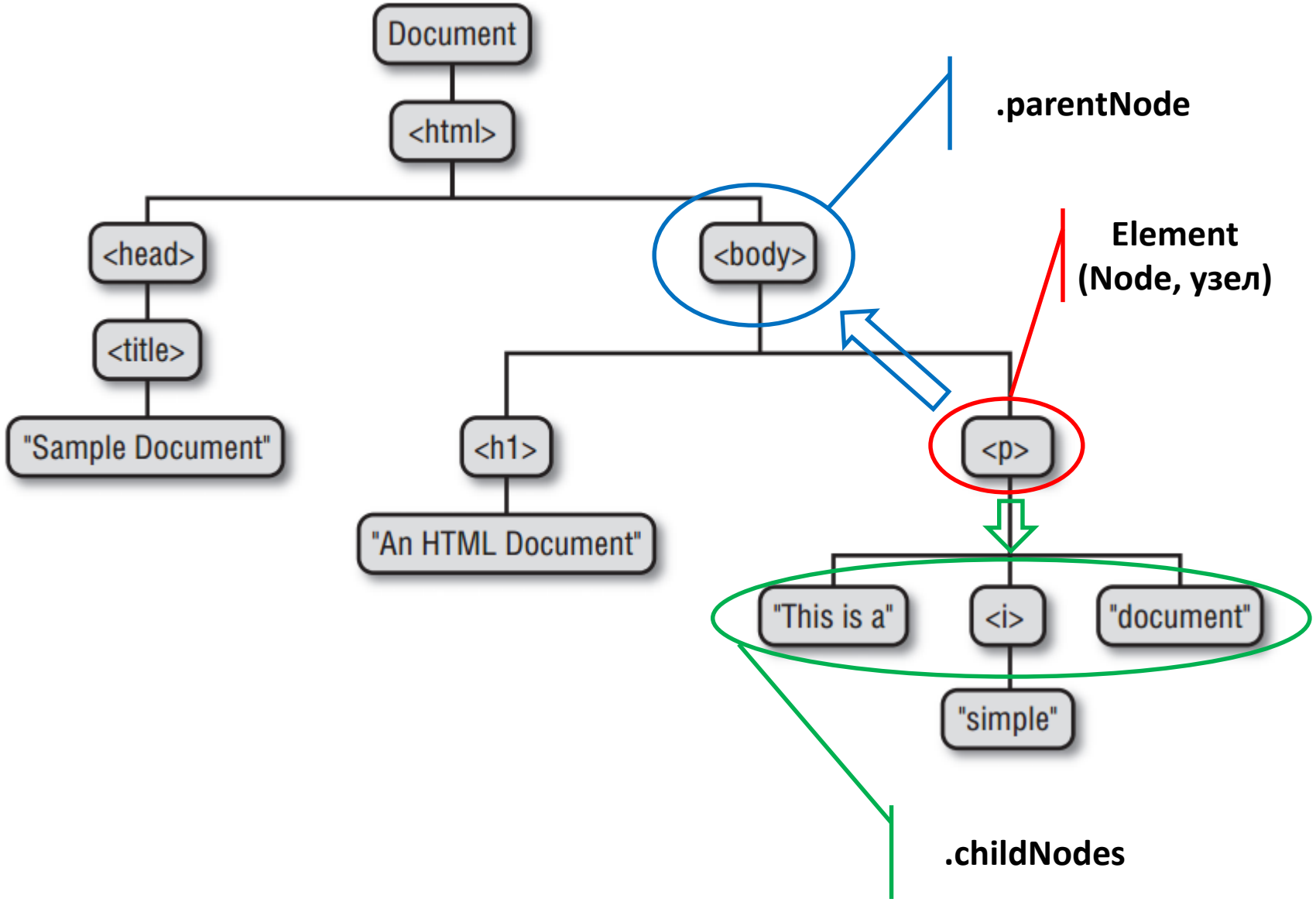
.id
.innerHTML
.className
.classList[…]
.attributes[…]
.style { … }

.onclick
.ondblclick
.onmouseenter

.appendChild()
.insertBefore()
.remove()
.insertAdjacentHTML()
.insertAdjacentElement()
.insertAdjacentText()
…

Также среди **свойств** объекта (**элемента, тега**) есть те которые позволяют управлять содержимым (**атрибутами**, **стилями**) или подпиской на **событиями**, а также ряд методов позволяющих добавлять/удалять элементы, и искать вложенные элементы.

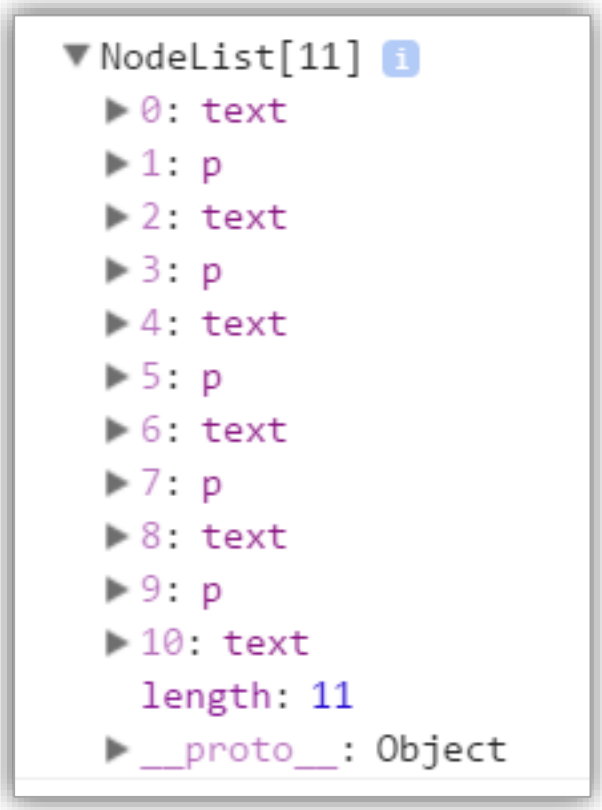
DOM – Document Object Model



window.document
корень дерева документа (globalThis)

window.document.childNodes (или **children**)– массив с тегами верхнего уровня (т.е. **html** и **doctype**).

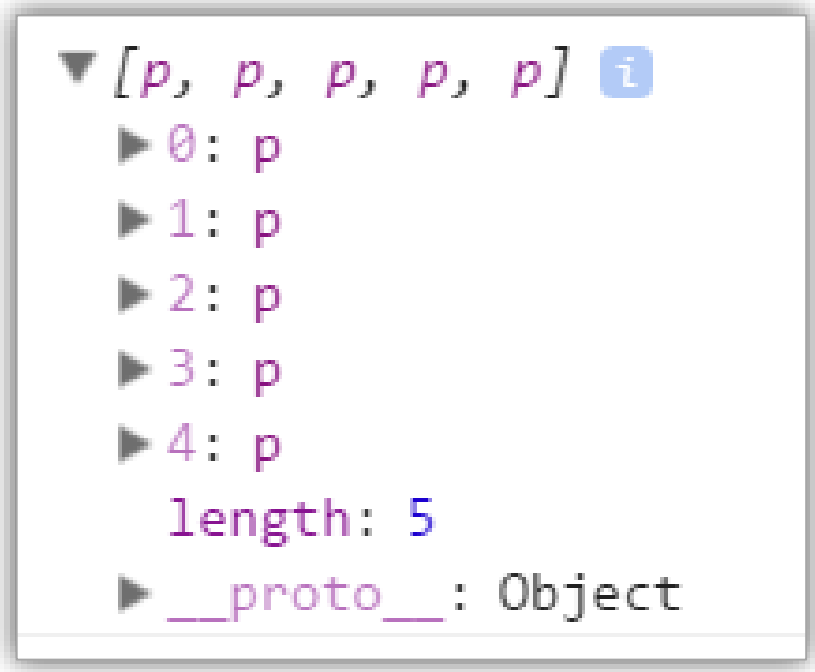
.children vs .childNodes



.childNodes

```
<div>
    <p>Text #1</p>
    <p>Text #2</p>
    <p>Text #3</p>
    <p>Text #4</p>
    <p>Text #5</p>
</div>
```

Свойство **.children** – тоже что и **.childNodes** но без «текстовых фрагментов»



.children

3. Когда выполняется код в теге <script> ?

JavaScript в HTML

<script></script>

Тег скрипт может быть размещен **в любом месте HTML-документа**, с помощью него можно либо непосредственно писать **JavaScript-код**, либо подключать внешний файл с кодом. Однако….

JavaScript в HTML

```
<script>
    abc.style.color     = "red";
    abc.style.border    = "3px dotted green";
    abc.style.textAlign = "center";
    abc.innerHTML       = "Hello world!!!";
</script>
<h1 id="abc"></h1>
```

```
<h1 id="abc"></h1>
<script>
    abc.style.color     = "red";
    abc.style.border    = "3px dotted green";
    abc.style.textAlign = "center";
    abc.innerHTML       = "Hello world!!!";
</script>
```

Код из тега **script** выполняется в тот момент когда браузер дойдёт до тега, если к этому моменту браузер еще не успел обработать разметку, то нашему коду не с чем будет работать.

JavaScript в HTML

Разрешить это неудобство (с выполнением кода сразу, а не когда страница полностью загрузится) можно разными способами, например:

1. Разместить весь код в конце документа;
2. Разместить весь код во внешнем файле и подключить его с атрибутом **defer**;
3. Использовать события **onLoad** или **onDOMContentLoaded** (эти варианты мы рассмотрим детальнее когда будет говорить о событиях).

```
<script defer src="scripts/async.js"></script>
```

*Атрибут **defer** откладывает выполнение скрипта до тех пор, пока вся страница не будет загружена полностью. Работает только для внешних (подключаемых) файлов.*

4. Как добраться (найти) до тега?

Теги у которых есть атрибут id доступны сразу как переменные ссылкой на объект

*Но только если **id** состоит из допустимых для имён переменных в **JavaScript** символов.*

Поиск элементов в документе

Выбор элемента с которым проводить манипуляции самая часто выполняемая операция в JS.

Выбор элемента по атрибуту id:

```
document.getElementById("some_id");
```

Возвращает один элемент атрибут (свойство) ***id*** *равно «**some_id**». Если такого элемента нет в документе, то возвращается **null**.*

Подробнее: https://learn.javascript.ru/searching-elements-dom

Поиск элементов в документе

Выбор элементов по названию тега:

```
document.getElementsByTagName("tag_name");
```

Выбор элементов по атрибуту name:

```
document.getElementsByName("attr_name");
```

Выбор элементов по атрибуту class:

```
document.getElementsByClassName("class_name");
```

Все эти функции возвращают псевдомассив с теми элементами которые подошли под условие.

Подробнее: https://learn.javascript.ru/searching-elements-dom

Поиск элементов в документе

Выбор всех элементов которые соответствуют CSS селектору:

```
document.querySelectorAll("css_selector");
```

Возвращает псевдомассив с теми элементами которые подошли под условие css-селектора.

```
document.querySelector("css_selector");
```

*Возвращает первый найденный элемент который подошел под условие css-селектора (или **null** если ничего не найдено).*

Подробнее: https://learn.javascript.ru/searching-elements-dom

Вложенный поиск, т.е. поиск в результатах поиска

```
54
55      let result_1 = document.querySelectorAll('p');
56
57      let columnOne = document.getElementById('column-one');
58      let result_2  = columnOne.querySelectorAll('p');
59
60      console.log('Result 1:', result_1);
61      console.log('Result 2:', result_2);
62
```

```
Result 1: ▶ NodeList(7) [p, p, p#special, p, p, p, p]
Result 2: ▶ NodeList(3) [p, p#special, p]
```

Функции поиска элементов можно применять к любому существующему элементу, а не только к документу. Когда функция поиска применяется к конкретному элементу, то поиск осуществляется среди его потомков.

«Живые» и статические коллекции

```
54
55      let result_1    = document.getElementsByTagName('p');
56      let result_2    = document.querySelectorAll('p');
57
58      special.remove();
59
60      console.log('Result 1:', result_1);
61      console.log('Result 2:', result_2);
62
```

```
Result 1: ▶ HTMLCollection(6) [p, p, p, p, p, p]
Result 2: ▶ NodeList(7) [p, p, p#special, p, p, p, p]
```

Живые (**Live**) коллекции изменяют свой состав в зависимости от изменений в документа. Статические (**Static**) коллекции не изменяют свой состав после формирования.

Живые и статические коллекции

.querySelector() и .querySelectorAll()

Возвращают статические коллекции, т.е. «слепок» на момент вызова функции.

.getElementsBy…()

Возвращают живые коллекции, которые всегда актуальны. Т.е. массив с результатом работы этих функций всегда будет содержать актуальное количество результатов, что бы не происходило с документом.

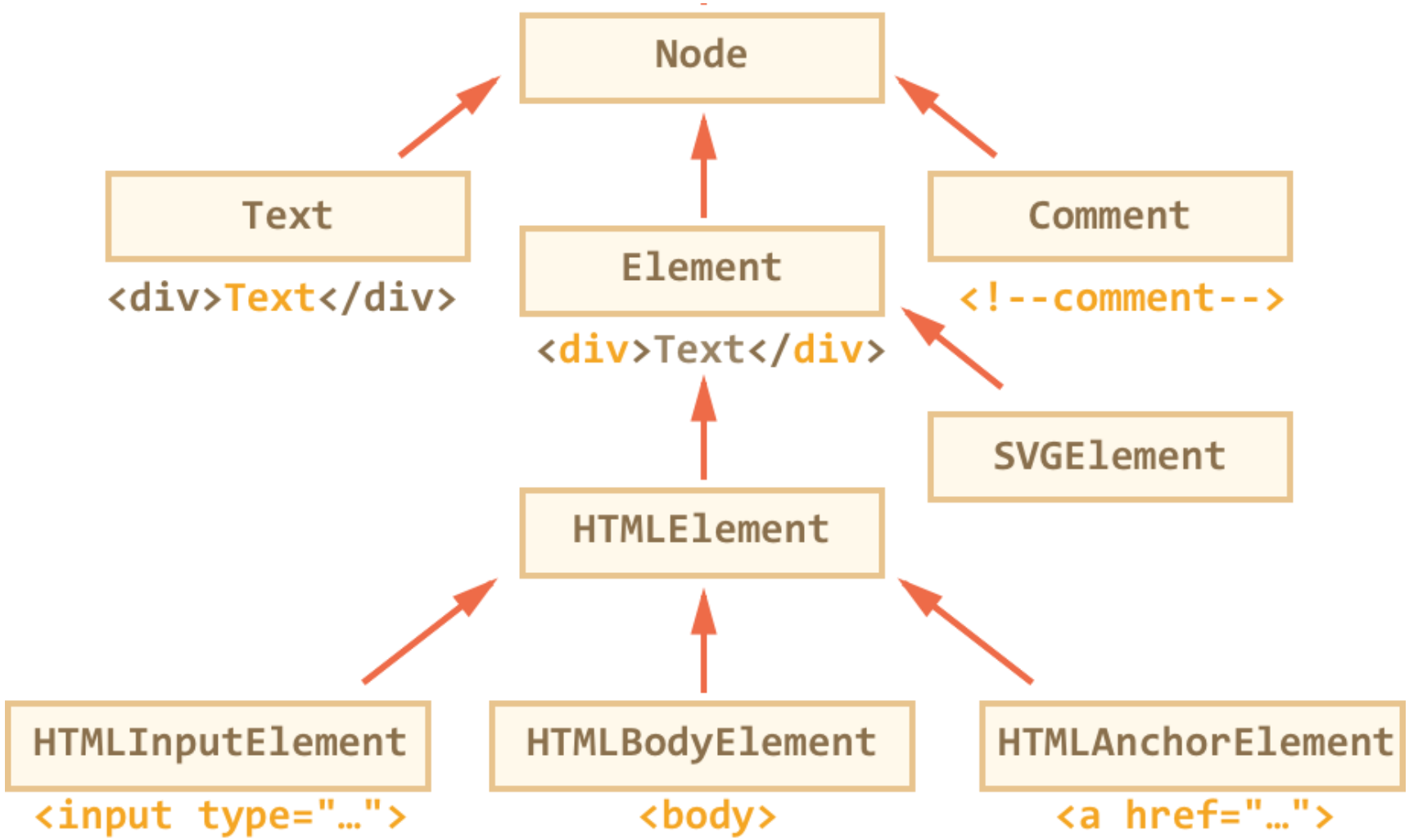
Подробнее: https://learn.javascript.ru/searching-elements-dom

С живыми коллекциями нужно быть осторожным *в том случае если вы перебираете её в цикле и изменяете её состав.*

```
54
55      let result  = document.getElementsByTagName('p');
56
57      result = [...result];
58
59      console.log('Result:', result);
60
```

Однако, и живую и статическую коллекцию можно конвертировать в классический массив.

Типы объектов в иерархии документа



Подробнее: https://learn.javascript.ru/basic-dom-node-properties

5. Как изменить тег?

Свойство .innerHTML хранит содержимое тега

Свойство .**innerHTML** – можно не только считывать но и устанавливать. Изменение свойства .**innerHTML** – автоматически влечёт перерисовку документа.

Полезные свойства элементов

.className *– свойство содержит полный список всех классов которые присвоены тегу (одной строкой).*

.classList *– свойство содержит список всех классов которые присвоены тегу (в виде массива).*

.classList.add('cat') *– метод добавляет класс к тегу (если есть другие классы то они остаются).*

.classList.remove('cat') *– метод удаляет класс у тегу (если есть другие классы то они не затрагиваются).*

.classList.toggle('cat') *– метод удаляет класс у тегу, если он есть, или добавляет класс, если его нет.*

.classList.contains('cat') *– метод проверяет наличие у тега заданного класса (возвращает true/false).*

.style *– свойство определяющее объект со всеми поддерживаемыми браузером стилевые свойства (CSS).*

.attributes *– хранит коллекцию с атрибутами тега.*

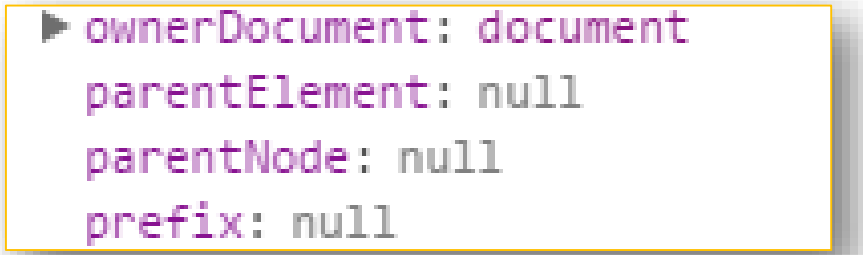
Подробнее: https://learn.javascript.ru/styles-and-classes

6. Как удалить тег?

Удаление элементов из дерева документа

```
54
55      let tag = document.getElementById('special');
56
57      tag.remove();
58
59      console.dir(tag);
60
```

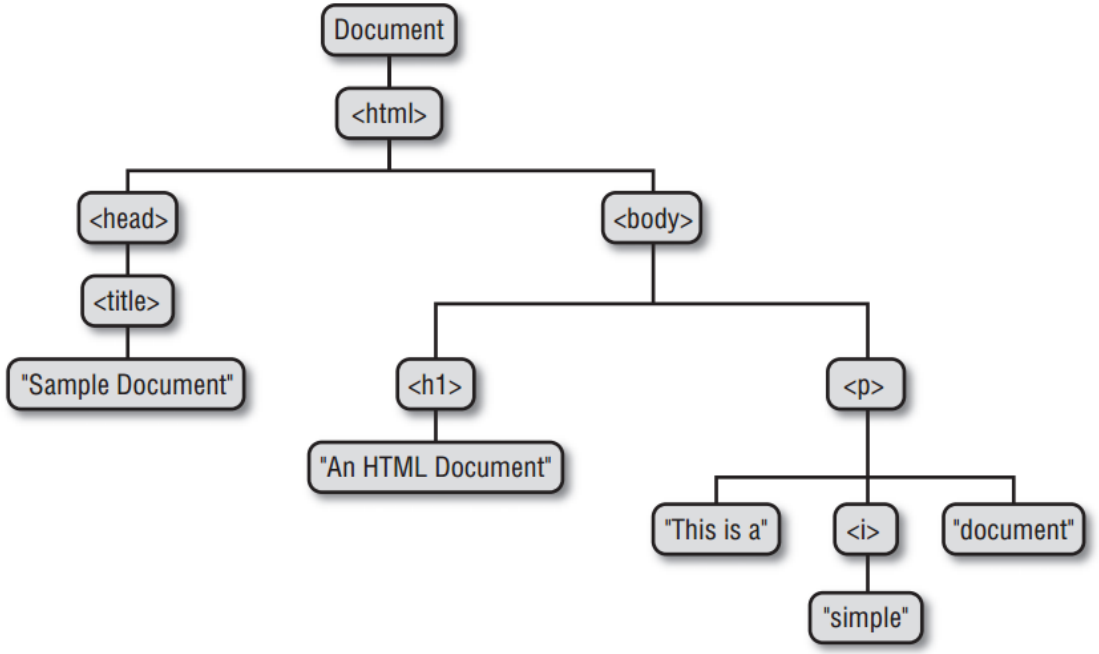
Удалить элемент из дерева документа можно вызывав у него метод **.remove()**, при этом все его дочерние элементы также исчезнут со странице. Однако сам объект-тег не уничтожается. Его можно использовать в дальнейшем.



7. Как создать и добавить тег?

Добавление новых элементов к дереву документа

Вставить новый элемент в документ, можно прикрепив его к какому-либо существующему элементу. Т.е. прикрепить его к родительскому элементу (другими словами: сделать его дочерним для существующего элемента).



Добавление новых элементов к дереву документа

Простейший вариант: просто добавить текстовую строку с нужными данным к свойству **.innerHTML**. Однако это не самый удобный вариант.

Добавление новых элементов к дереву документа (первое поколение)

document.createElement() – создаёт новый элемент (по имени тега). Этот элемент, после создания, еще не включен в дерево. Но его свойства уже можно изменять.

.appendChild() – добавляет элемент к существующему, в качестве последнего потомка. Может быть вызвана для любого существующего тега (даже если он не входит в дерево – другими словами можно формировать ветку еще до того как «присоединять» её к дереву).

.insertBefore() – добавляет элемент в качестве дочернего, при этом позволяет указать перед каким из, уже существующих, потомков новый элемент должен быть размещён.

Подробнее: https://learn.javascript.ru/modifying-document

Добавление новых элементов к дереву документа (второе поколение)

```
<!-- beforebegin -->
<p>
  <!-- afterbegin -->
  foo
  <!-- beforeend -->
</p>
<!-- afterend -->
```

*Варианты позиции для методов группы **.insertAdjacent**...()*

`tag.insertAdjacentElement(position, element)`

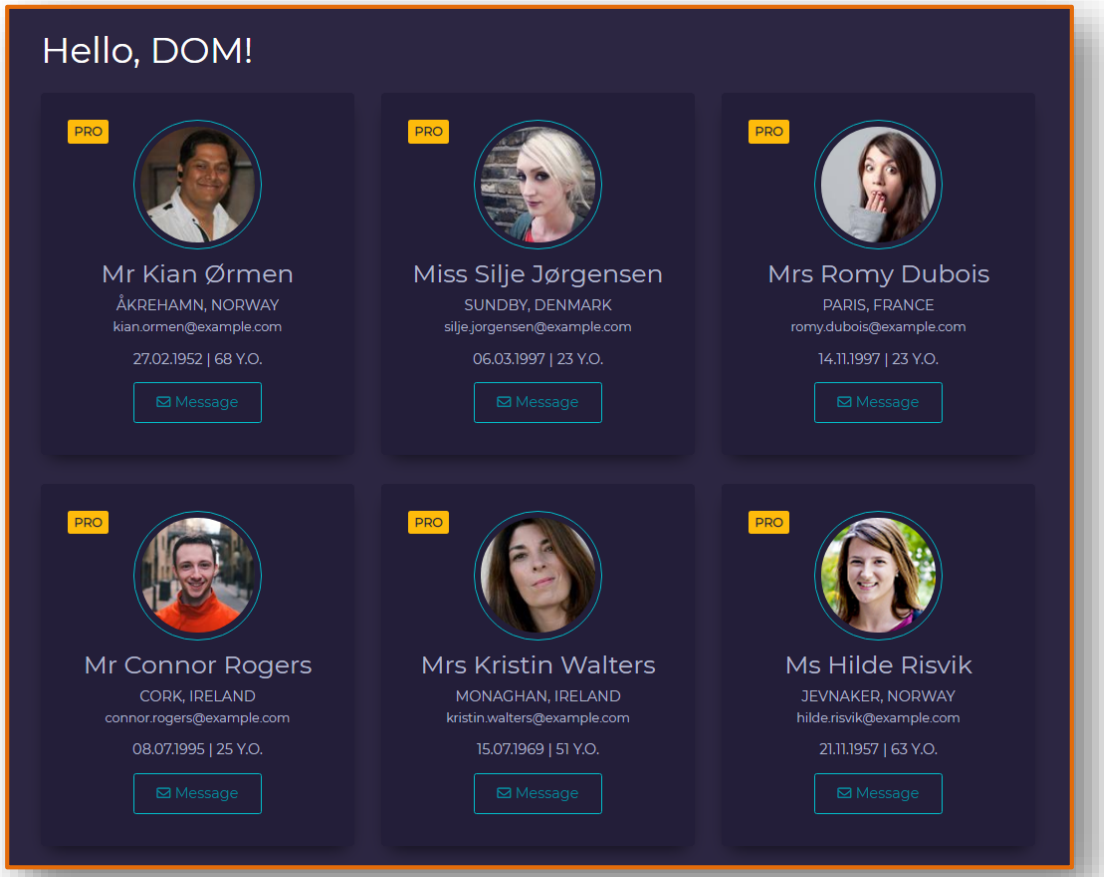
*добавляет **элемент** к **существующему**, в указанную **позицию**.*

Подробнее: https://learn.javascript.ru/modifying-document

*Также существуют методы **tag.insertAdjacentHTML() и tag.insertAdjacentText()***

8. Немного практики

DOM на практике



В репозитории занятия воспользуйтесь шаблоном: **./src/demo-example-2**

Выведем в подготовленную разметку данные пользователей полученные от сервиса **https://randomuser.me/**

9. Export/Import (ES Modules)

Процедура экспорта/импорта модулей (ES Modules)

main.js

```
import def from './lib.js';

import { pi as Pi, sum, config } from './lib.js';

console.log(def, Pi, sum, config);
```

```
<script src="./main.js" type='module'></script>
```

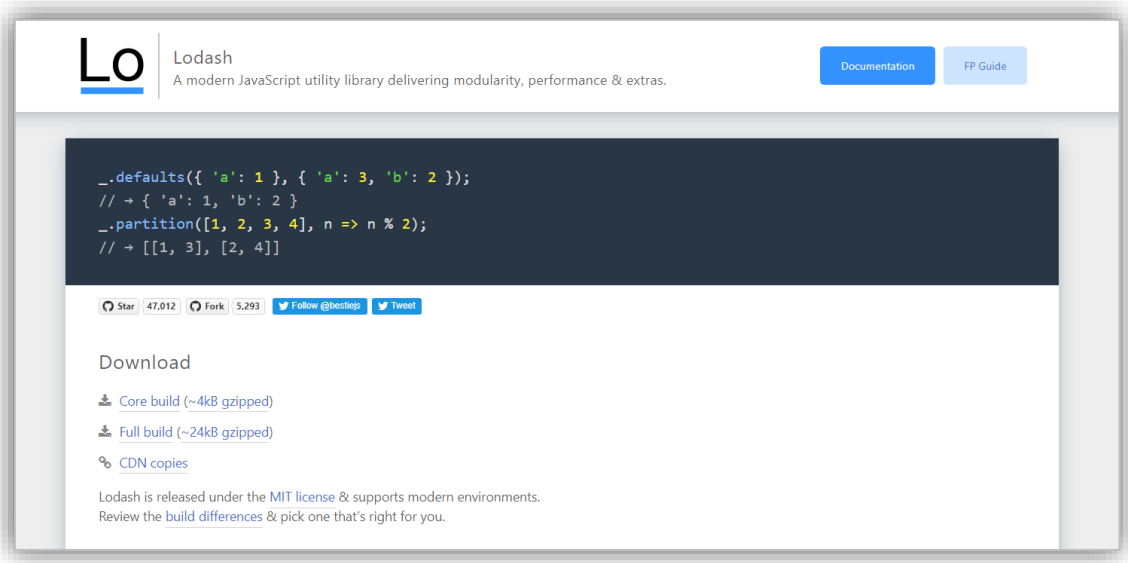
lib.js

```
const pi = 3.14;

function sum(a, b) {
    return a + b;
}

let config = {
    enable: true,
    count: 42,
    id: 'HX883'
}

export { pi, sum , config };

export default config;
```

Директивы **export/import** по сути позволяют подключать сторонние (специальным образом подготовленные) *js-файлы* (**ES-модули**) с кодом непосредственно из *js-кода*. Для работы этого механизма первый файл (в котором импортируются другие) должен быть подключен с атрибутом **type='module'**.

https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Statements/export
https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Statements/import

Lodash – библиотека для работы с данными



У библиотеки **lodash** есть версия с поддержкой **ES-Modules** и без неё.

https://www.npmjs.com/package/lodash

https://www.npmjs.com/package/lodash-es

https://lodash.com/

К следующему занятию будет полезно почитать о…

К следующему занятию…

Обработка событий (DOM Events)

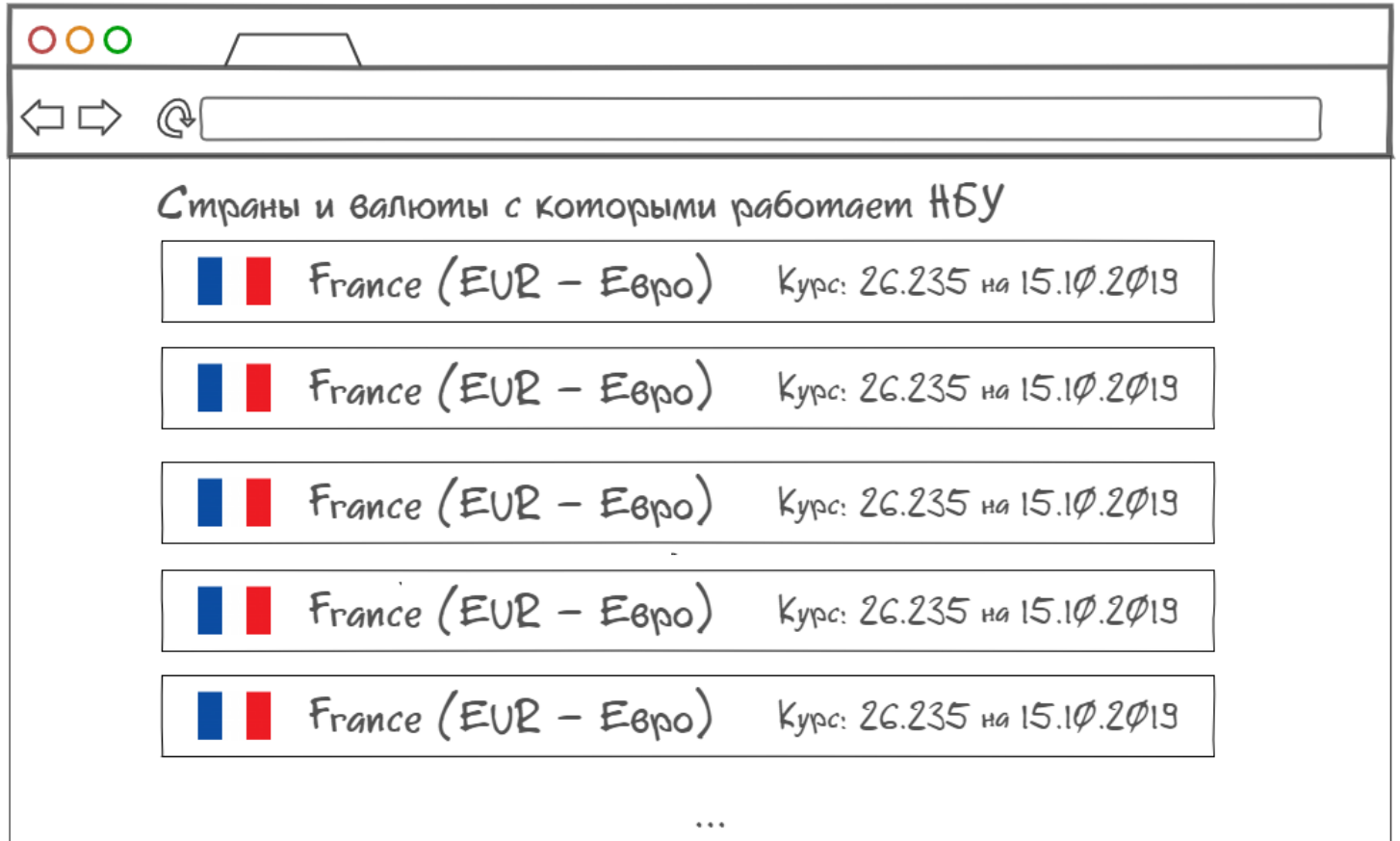
Домашнее задание

/сделать

Домашнее задание #G.1

Воспользуйтесь API дающее информацию о странах мира:
https://restcountries.eu/rest/v2/all

Так же воспользуйтесь API НБУ по курсам валют:
https://bank.gov.ua/NBUStatService/v1/statdirectory/exchange?json



Выведите в разметку **перечень стран** с валютами которых работает НБУ (если несколько стран имеют общую валюту - **выводите все эти страны**, пример: зона Евро, или страны использующие USD). Пример разметки на wireframe. **Разметку необходимо подготовить)))**