Лабораторная работа №3 (часть1)

“Работа с DOM. События”

Реализовать функции согласна своего варианта.

**Вариант 1**

**Задание 1.**

Функция должна перебрать все дочерние элементы элемента where и вернуть массив, состоящий из тех дочерних элементов следующим соседом которых является элемент с тегом P (с рекурсией)

@param {Element} where - где искать

@return {Array<Element>}

\* @example

для html '<div></div><p></p><a></a><span></span><p></p>'

функция должна вернуть: [div, span]

т.к. следующим соседом этих элементов является элемент с тегом P

*function findAllPSiblings(where) {*

*}*

**Задание 2.**

Функция должна перебрать все дочерние узлы элемента where и удалить из него все текстовые узлы. Без рекурсии! Будьте внимательны при удалении узлов, можно получить неожиданное поведение при переборе узлов

@param {Element} where - где искать

@example

после выполнения функции, дерево <div></div>привет<p></p>loftchool!!!

должно быть преобразовано в <div></div><p></p>

*function deleteTextNodes(where) {*

*}*

**Задание 3.**

Функция должна добавить такой обработчик кликов к элементу target который сработает только один раз и удалится. Постарайтесь не создавать глобальных переменных

@param {Element} target - элемент, на который нужно добавить обработчик

@param {function} fn - обработчик

*function once(target, fn) {*

*}*

**Вариант 2**

**Задание 1.**

Функция должна вставлять элемент what в начало элемента where

@param {Element} what - что вставлять

@param {Element} where - куда вставлять

*function prepend(what, where) {*

*}*

**Задание 2.**

Функция должна перебрать все дочерние узлы элемента where и удалить из него все текстовые узлы. С рекурсией! Будьте внимательны при удалении узлов, можно получить неожиданное поведение при переборе узлов

@param {Element} where - где искать

@example

после выполнения функции, дерево <span> <div> <b>привет</b> </div> <p>loftchool</p> !!!</span>

должно быть преобразовано в <span><div><b></b></div><p></p></span>

*function deleteTextNodesRecursive(where) {*

*}*

**Задание 3.**

Функция должна добавлять к target обработчик события eventName, который должен отменять действие по умолчанию

@param {string} eventName - имя события, для которого нужно удалить обработчик

@param {Element} target - элемент, на который нужно добавить обработчик

*function skipDefault(eventName, target) {*

*}*

**Вариант 3**

**Задание 1.**

Функция должна создать элемент с тегом A, установить значение для атрибута href и вернуть получившийся элемент

@param {string} hrefValue - значение для атрибута href

@return {Element}

*function createAWithHref(hrefValue) {*

*}*

**Задание 2.**

Необходимо собрать статистику по всем узлам внутри элемента root и вернуть ее в виде объекта

Статистика должна содержать:

- количество текстовых узлов

- количество элементов каждого класса

- количество элементов каждого тега

Для работы с классами рекомендуется использовать свойство classList

Постарайтесь не создавать глобальных переменных

@param {Element} root - где собирать статистику

@return {{tags: Object<string, number>, classes: Object<string, number>, texts: number}}

@example

для html <div class="some-class-1"><b>привет!</b> <b class="some-class-1 some-class-2">loftschool</b></div>

должен быть возвращен такой объект:

{

tags: { DIV: 1, B: 2},

classes: { "some-class-1": 2, "some-class-2": 1 },

texts: 3

}

*function collectDOMStat(root) {*

*}*

**Задание 3.**

Функция должна удалять обработчик fn события eventName у элемента target

@param {string} eventName - имя события, для которого нужно удалить обработчик

@param {Element} target - элемент, у которого нужно удалить обработчик

@param {function} fn - обработчик

*function removeListener(eventName, target, fn) {*

*}*

**Вариант** 4

**Задание 1.**

Функция должна создать элемент с тегом DIV, поместить в него текстовый узел и вернуть получившийся элемент

@param {string} text - текст, который необходимо поместить в div

@return {Element}

*function createDivWithText(text) {*

*}*

**Задание 2.**

Функция должна отслеживать добавление и удаление элементов внутри элемента where. Как только в where добавляются или удаляются элемента, необходимо сообщать об этом при помощи вызова функции fn со специальным аргументом. В качестве аргумента должен быть передан объек с двумя свойствами:

- type: типа события (insert или remove)

- nodes: массив из удаленных или добавленных элементов (а зависимости от события)

Отслеживание должно работать вне зависимости от глубины создаваемых/удаляемых элементов

Рекомендуется использовать MutationObserver

@param {Element} where - где отслеживать

@param {function(info: {type: string, nodes: Array<Element>})} fn - функция, которую необходимо вызвать

@example

если в where или в одного из его детей добавляется элемент div то fn должна быть вызвана с аргументов:

{

type: 'insert',

nodes: [div]

}

если из where или из одного из его детей удаляется элемент div то fn должна быть вызвана с аргументов:

{

type: 'remove',

nodes: [div]

}

*function observeChildNodes(where, fn) {*

*}*

**Задание 3**

Функция должна добавлять обработчик fn события eventName к элементу target

@param {string} eventName - имя события, на которое нужно добавить обработчик

@param {Element} target - элемент, на который нужно добавить обработчик

@param {function} fn - обработчик

*function addListener(eventName, target, fn) {*

*}*