**Отчёт по лабораторной работе №2**

**Ответы на задания:**

**1.**

var element = document.querySelector("table tr:nth-child(8) td:nth-child(7)");

var link = element.querySelector("a[href='#delete']");

link.click();

console.log(link);

**2.**

var checkbox1 = document.querySelector("input[type='checkbox']");

var checkbox2 = document.querySelectorAll("input[type='checkbox']")[1];

checkbox1.click();

checkbox2.click();

**3.**

var dropdown = document.querySelector("select#dropdown");

dropdown.value = "1";

setTimeout(() => {

  dropdown.value = "2";

}, 5000);

**4.**

var modal = document.querySelector("#modal");

if (modal){

    modal.style.display = "none";

}

**5.**

var usernameInput = document.querySelector("input[name='username']");

if (usernameInput) {

  usernameInput.value = "tomsmith";

}

var passwordInput = document.querySelector("input[name='password']");

if (passwordInput) {

  passwordInput.value = "SuperSecretPassword!";

}

var submitButton = document.querySelector("button[type='submit']");

if (submitButton) {

  submitButton.click();

}

**6.**

function waitForElement(selector) {

  return new Promise((resolve, reject) => {

    const element = document.querySelector(selector);

    if (element) return resolve(element);

    const observer = new MutationObserver(() => {

      const el = document.querySelector(selector);

      if (el) {

        observer.disconnect();

        resolve(el);

      }

    });

    observer.observe(document.body, { childList: true, subtree: true });

  });

}

function waitForButton(label) {

  return new Promise((resolve, reject) => {

    const findBtn = () =>

      [...document.querySelectorAll("button")].find(

        (btn) => btn.textContent.trim() === label

      );

    var btn = findBtn();

    if (btn) return resolve(btn);

    const observer = new MutationObserver(() => {

      btn = findBtn();

      if (btn) {

        observer.disconnect();

        resolve(btn);

      }

    });

    observer.observe(document.body, { childList: true, subtree: true });

  });

}

function waitForAttributeRemoval(selector, attribute) {

  return new Promise((resolve, reject) => {

    const el = document.querySelector(selector);

    if (el && !el.hasAttribute(attribute)) return resolve(el);

    const observer = new MutationObserver(() => {

      const elNow = document.querySelector(selector);

      if (elNow && !elNow.hasAttribute(attribute)) {

        observer.disconnect();

        resolve(elNow);

      }

    });

    observer.observe(document.body, { childList: true, subtree: true, attributes: true });

  });

}

async function task1() {

  const checkbox = await waitForElement("input[type='checkbox']");

  checkbox.click();

  await new Promise(r => setTimeout(r, 2000));

  const removeBtn = await waitForButton("Remove");

  removeBtn.click();

  const addBtn = await waitForButton("Add");

  addBtn.click();

  const newCheckbox = await waitForElement("input[type='checkbox']");

  newCheckbox.click();

}

async function task2() {

  const enableBtn = await waitForButton("Enable");

  enableBtn.click();

  const textInput = await waitForAttributeRemoval("input[type='text']", "disabled");

  textInput.value = "Hello world!";

  const disableBtn = await waitForButton("Disable");

  disableBtn.click();

}

async function run() {

  await Promise.all([task1(), task2()]);

}

run();

**7.**

var slider = document.querySelector('input[type="range"]');

if (slider) {

  slider.value = 0;

  slider.dispatchEvent(new Event("change", { bubbles: true }));

  var current = 0;

  const max = 5;

  const step = 0.5;

  const interval = setInterval(() => {

    current += step;

    slider.value = current;

    slider.dispatchEvent(new Event("change", { bubbles: true }));

    if (current >= max) {

      clearInterval(interval);

      slider.dispatchEvent(new Event("change", { bubbles: true }));

    }

  }, 1000);

}

**8.**

const captions = document.querySelectorAll('.figcaption');

captions.forEach(el => {

    el.style.display = 'block';

  });

setTimeout(() => {

    const link = captions[1].querySelector("a");

    link.click();

}, 10000);

**9.**

function findHeaderBySpan(table, text) {

  return Array.from(table.querySelectorAll("th"))

    .find(th => {

      const span = th.querySelector("span");

      return span && span.textContent.trim() === text;

    });

}

const table1 = document.querySelector("#table1");

const table2 = document.querySelector("#table2");

[table1, table2].forEach(tbl => {

const th = findHeaderBySpan(tbl, "First Name");

th.click();

});

setTimeout(() => {

[table1, table2].forEach(tbl => {

    const th = findHeaderBySpan(tbl, "Due");

    th.click();

});

}, 5000);

**10.**

const newText = "Helloooooooooooooooooooooo";

const div = document.querySelector('#sibling-41\\.1');

div.childNodes[0].nodeValue = newText;

const cell = document.querySelector('tr.row-41 td.column-1');

cell.textContent = newText;

**Ответы на вопросы:**

1. Расскажите про команду document.querySelector

document.querySelector – метод в JavaScript, который ищет первый элемент в DOM, подходящий под заданный селектор.

Селектор пишется как в CSS: #value – поиск по id, .value – поиск по классу, [attr=value] – поиск по атрибуту и т.д.

2. Расскажите про команду $x

$x – шорткат в консоли разработчика браузера, осуществляющий поиск элемента по селектору аналогично document.querySelector, но с использованием синтаксиса xPath, например: $x(“div[@id=’value’]”)

3. Расскажите про команду document.evaluate

document.evaluate – метод JavaScript, используемый для работы с запросами xPath. Имеет параметры:

xPathExpression – выражение xPath

contextNode – элемент DOM, в котором происходит запрос namespaceResolver – аргумент, используемый для обработки других пространств имён (например в SVG или XML)

resultType – тип возвращаемого результата

result – переменная с результатом.