# ЛАБОРАТОРНАЯ РАБОТА №3
# JAVASCRIPT

**Цель работы:** получить навык разработки с использованием языка программирования JavaScript

**Задачи:**

1. Разработать JS-скрипт, реализующий действие на странице в браузере согласно варианту
2. Добавить скрипт к странице.

**Результатами работы являются:**

1. Разработанный скрипт.
2. Подготовленный отчет.

## ИСПОЛЬЗОВАНИЕ JAVASCRIPT В HTML

JS – скриптовый язык. Скрипты могут встраиваться в HTML и выполняться браузером при загрузке. Также JavaScript может выполняться не только в браузере, но и на сервере или на любом другом устройстве, которое имеет специальную программу, называющуюся «движком» (виртуальной машиной) JavaScript.

JavaScript может как встраиваться в HTML, так и быть вынесенным в отдельный .js файл (рекомендуемый подход). Пример использования JS:

```
<!DOCTYPE HTML>
<html>
    <head>
          <script src="/path//someScript.js"></script>
    </head>
    <body>
          <p>Перед скриптом...</p>
          <script>
                alert( 'Привет, мир!' );
          </script>
           <p>...После скрипта.</p>
    </body>
</html>
```

**Объявление переменных в JS**

Область видимости переменных var ограничивается либо функцией, либо, если переменная глобальная, то скриптом. Такие переменные доступны за пределами блока. Объявления (инициализация) переменных var производится в начале исполнения функции (или скрипта для глобальных переменных). Для более «традиционного» объявления переменных рекомендуется использовать ключевое слово let. В JS, как и в PHP динамическая типизация.

Типы данных:

– Числовой тип данных (number) представляет как целочисленные значения, так и числа с плавающей точкой.

Кроме обычных чисел, существуют так называемые «специальные числовые значения», которые относятся к этому типу данных: Infinity, -Infinity и NaN. не может содержать числа больше, чем (2^53-1) (т. е. 9007199254740991), или меньше, чем -(2^53-1)

- BigInt. Чтобы создать значение типа BigInt, необходимо добавить n в конец числового литерала:
  const bigInt = 1234567890123456789012345678901234567890n;
- Булевый тип (boolean) может принимать только два значения: true (истина) и false (ложь).
- Строка
- Специальное значение null не относится ни к одному из типов, описанных выше. Оно формирует отдельный тип, который содержит только значение null:
- Специальное значение undefined также стоит особняком. Оно формирует тип из самого себя так же, как и null. Оно означает, что «значение не было присвоено». Если переменная объявлена, но ей не присвоено никакого значения, то её значением будет undefined
- Объекты – используется для коллекций данных и для объявления более сложных сущностей. Объявляются объекты при помощи фигурных скобок {...}.
- Тип symbol (символ) используется для создания уникальных идентификаторов в объектах

Для определения типа переменной можно воспользоваться оператором typeof, у него есть две синтаксические формы:

- Синтаксис оператора: typeof x.
- Синтаксис функции: typeof(x).

Однако, при использовании typeof есть несколько особенностей:

- При применении для функции будет возвращено значение function, несмотря на то что такого типа нет и функции представляют из себя объекты.

– При применении для значения null будет возвращено значение object, хотя null представлят из себя отдельный тип данных.

Также в JS есть ряд объектов-прототипов со своим набором методов для работы с данными: массив, словарь и др.

В JS используется прототипное наследование, работающее по следующему принципу: при обращении к полю объекта это поле ищется в самом объекте, если оно не находится, то поиск продолжается в объекте, указанном в поле prototype и т.д. по цепочке прототипов пока не будет найдено нужное поле или пока не найдется прототип, у которого не будет указано поле prototype.

**Функции**

Функции в JS – это значения. Они могут быть присвоены, скопированы или объявлены в другом месте кода. Есть 2 способа задания функций: декларативный и функциональный:

```
//Function Declaration
function sayHi() {
    alert( "Привет" );
}

//Function Expression
let sayHi = function() {
    alert( "Привет" );
};
```

В отличие от других С-подобных языков ключевое слово this в JS вычисляется во время выполнения кода и зависит от контекста, т.е. объекту (а, значит, и функции) можно в любой момент выполнения задать необходимый контекст, к которому можно будет обратиться с помощью ключевого слова this. Задать контекст можно с помощью метода bind(context, [params]), либо можно вызвать функцию с определенным контекстом с помощью метода call(context, [params]).

В JS есть возможность создания анонимных функций, для этого можно использовать arrow function (стрелочные функции), которые

представляют из себя набор параметров знак => и само тело функции. Важно: у стрелочных функций нет контекста! Кроме того, можно создать самовызываемую функцию, для этого необходимо обернуть функцию в (), а затем указать ее входные параметры. Пример самовызываемой стрелочной функции:

```
((val) => console.log(val))("Привет")
```

**Работа с DOM в JS**

После загрузки HTML/XML документы представляются в браузере в виде DOM-дерева (Document Object Model) – JS объекта, которым удобно управлять и с которым удобно производить изменения в отличие от текстового формата HTML/XML. Теги становятся узлами-элементами и формируют структуру документа. Текст становится текстовыми узлами. После построения DOM бразуер будет работать именно с построенным объектом, а не с самим HTML.

В JS коде есть возможность получать доступ к элементам DOM для задания свойств и изменения самой структуры DOM. Для доступа к DOM используется глобальный объект document, например document.body – объект для тега <body>.

Между узлами DOM можно перемещаться, а также можно производить поиск элементов:

- querySelector(selector) возвращает первый элемент, который соответствует одному или более CSS селекторам. Если совпадения не будет, то он вернет null.
- querySelectorAll() возвращает все элементы, которые подходят под указанный CSS селектор.Подходящие элементы возвращаются в виде NodeList объекта, который будет пустым в случае того, если не будет найдено совпадений.
- getElementById – получение элемента по значению id. В качестве результата метод getElementById возвращает ссылку на объект типа Element или значение null, если элемент с указанным идентификатором не найден. Метод getElementById имеется только у объекта document/

– getElementsByName – получение элементов по значению атрибута name. Аналогично предыдущему, метод имеется только у объекта document.
– getElementsByClassName – получение списка элементов по именам классов. Этот метод позволяет найти все элементы с указанными классами во всём документе или в некотором элементе.
– getElementsByTagName – получение элементов по имени тега.

Найдя интересующие узлы DOM-дерева, с ними можно производить различные манипуляции:

1. Получение и задание содержимого:
   – innerHTML – внутреннее HTML-содержимое узла-элемента. Можно изменять.
   – outerHTML – полный HTML узла-элемента. Запись в elem.outerHTML не меняет elem. Вместо этого она заменяет его во внешнем контексте.
   – textContent – текст внутри элемента: HTML за вычетом всех тегов. Запись в него помещает текст в элемент, при этом все специальные символы и теги интерпретируются как текст.
2. Получение и устанавка значений аттрибутов:
   – elem.hasAttribute(name) – проверяет наличие атрибута.
   – elem.getAttribute(name) – получает значение атрибута.
   – elem.setAttribute(name, value) – устанавливает значение атрибута.
   – elem.removeAttribute(name) – удаляет атрибут.
3. Создание элементы:
   – document.createElement(tag) – создаёт новый элемент с заданным тегом
   – document.createTextNode(text) – создаёт новый текстовый узел с заданным текстом
4. Добавление информации к узлам:
   – node.append(...nodes or strings) – добавляет узлы или строки в конец node.

- node.prepend(...nodes or strings) – вставляет узлы или строки в начало node.
- node.before(...nodes or strings) –- вставляет узлы или строки до node.
- node.after(...nodes or strings) –- вставляет узлы или строки после node.
- node.replaceWith(...nodes or strings) –- заменяет node заданными узлами или строками.

5. Удаление:
   - node.remove()
6. Клонирование:
   - elem.cloneNode(true)

**События**

Событие – это сигнал от браузера о том, что что-то произошло. Событию можно назначить обработчик, то есть функцию, которая сработает, как только событие произошло. Примеры:

```
<input value="Нажми меня" onclick="alert('Клик!')"
    type="button">

<input id="elem" type="button" value="Нажми меня!">
<script>
    elem.onclick = function() {
    alert('Спасибо');
};
</script>
element.addEventListener(event, handler[, options]);
```

По умолчанию для влоенных элементов применяется механизм всплытия: после обработки события целевым элементом событие передается вверх родительскому элементу и т.д. до верхнего элемента или до вызова метода event.stopPropagation(). Также можно включить механизм погружения (вначале вызываются обработчики родительских элементов, начиная с верхнего, потом целевого, а затем

повторяется всплытие. На каждом этапе можно получить фазу обработки события, обратившись к свойству event.eventPhase), для этого обработчику события необходимо указать свойство {capture: True}.

При обработке событий можно отключить поведение браузера по умолчанию (например, перезагрузку страницы после отправки формы) с помощью метода event.preventDefault(). Если обработчик назначен через on<событие> (не через addEventListener), то также можно вернуть false из обработчика.

**Загрузка скриптов**

При указании JS-скрипта на странице, рендеринг страницы приостановится до того момента пока не загрузится указанный скрипт. Иногда это нерационально, поскольку у пользователя будет либо пустая, либо недорисованная страница, хотя на ее внейшний вид скрипт может никак не влиять. Для асинхронной загрузки скрипта, т.е. загрузки «на фоне» без влияния на рендеринг страницы можно указать ключевое слово defer.

Для оптимизации загрузки страницы может потребоваться разбить скрипт на несколько составных частей. В таком случае все скрипты, загруженные с указанием ключевого слова defer, будут загружены последовательно. Если же скрипт абсолютно независим и его можно загружать асинхронно, то вместо defer нужно использовать ключевое слово async.

Скрипты можно загружить и с помощью JS, такие скрипты по умолчанию будут загружены асихронно:

```
let script = document.createElement('script’);
script.src = "/long.js";
document.body.append(script); // (*)
```

**Promisis**

Для асинхронного выполнения кода можно использовать механизм промисов (от англ. promise – обещание). Промис представляет из себя объект, выполняющий некое асинхронное действие и имеющий

переданными параметрами 2 функции: функцию, которую необходимо выполнить при успешном завершении промиса и функцию, которую необходимо выполнить при ошибке во время выполнения. Самый простой пример промиса:

```
let promise = new Promise(function(resolve, reject) {
    setTimeout(() =>
          reject(new Error("Whoops!")), 1000);
});
```

Кроме того, из промисов можно выстраивать цепочки, используя потребитель промиса then, который также возвращает промис:

```
new Promise(function(resolve, reject) {
    setTimeout(() => resolve(1), 1000);
}).then(function(result) {
    alert(result);
    return new Promise((resolve, reject) => {
          setTimeout(() => resolve(result * 2), 1000);
    });
})
```

Promise API:

- let promise = Promise.all([...промисы...]) – ожидает пока выполнятся все промисы в массиве.
- let promise = Promise.allSettled([...промисы...]) – ожидает успешного выполнения всех промисов в массиве
- let promise = Promise.race([...промисы...]) – ожидает пока выполнится хоть один из промисов в массиве.
- Promise.resolve(value) – возвращает успешный промис из значения (может быть необходимо в цепочке промисов). Является устаревшим.
- Promise.reject(value) – возвращает неуспешный промис из значения (может быть необходимо в цепочке промисов. Является устаревшим.

Однако, вместо использования .then() рекомендуется использовать ключевые слова async и await. У слова async один простой смысл: эта функция всегда возвращает промис. Значения других типов оборачиваются в завершившийся успешно промис автоматически.

Ключевое слово await заставит интерпретатор JavaScript ждать до тех пор, пока промис справа от await не выполнится. После чего оно вернёт его результат, и выполнение кода продолжится. Например:

```
async function f() {
    let promise = Promise((resolve, reject) => {
          setTimeout(() => resolve("Ok!"), 1000);
    let result = await promise;
}
```

**Работа с Canvas**

<canvas> — это HTML5 элемент, использующийся для рисования графики средствами JavaScript. В скрипте, получив контекст canvas и задав стили рисования (например, цвет, толщину линий, разделитель и др.) можно рисовать примитивы (линия, прямоугольник, дуга и др.):

```
<head>
    <meta charset="utf-8"/>
    <script type="application/javascript">
          function draw() {
                const canvas = document
                      .getElementById('canvas');
                if (canvas.getContext) {
                      const ctx = canvas.getContext('2d');
                      ctx.fillStyle = 'rgb(200, 0, 0)';
                      ctx.fillRect(5, 5, 20, 20);
                      ctx.fillStyle = 'rgba(0, 0, 90, 0.5)';
                      ctx.fillRect(30, 30, 50, 50);
                }
          }
    </script>
</head>
<body onload="draw();">
    <canvas id="canvas" width="150" height="150"></canvas>
</body>
```

Для рисования фигур необходимо вызвать метод beginPath, после чего можно указывать точки фигуры, с помощью метода stroke будет нарисована ломанная, с помощью метода fill можно нарисовать закрашенную фигуру:

```
ctx.beginPath();
ctx.moveTo(5 + i * 14, 5);
ctx.lineTo(5 + i * 14, 140);
ctx.stroke();
```

Для рисования дуг применяется метод arc, например, для рисования окружности:

```
ctx.arc(100, 100, 75, 0, getRadians(360));
```

Перед рисованием к фигурам можно применить преобразования поворота и сдвига:

```
ctx.rotate(angle);
ctx.translate(105,0);
```

Для очистки холста используется метод clearRect(), для очиски всего экрана можно воспользоваться:

```
ctx.clearRect(0, 0, canvas.width, canvas.height);
```

Для создания анимации необходимо с некоторой периодичности очищать экран и перерисовывать изображение, этого можно добиться, например, с помощью функции window.requestAnimationFrame(draw), в качестве параметра передается функция отрисовки (частота обновления хзависит от браузера, чаще всего 60 раз в секунду).

При отрисовке сложной сцены может потребоваться множество раз задавать одни и теже свойства рисования, чтобы этого избежать, свойства можно сохранять в стек с помощью метода ctx.save(), а затем восстанавливать с помощью метода ctx.restore();

## ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Разработать скрипт согласно варианту, полученному у преподавателя.

## ВАРИАНТЫ ЗАДАНИЙ

1. Разработать браузерную игру «Крестики-нолики».
2. Разработать браузерное приложение для подсчета интегралов задаваемых полиномиальных функций в указанных промежутках методами прямоугольников и трапеций.
3. Разработать браузерную игру «Змейка»
4. Разработать браузерное приложение для решения СЛАУ методом Гаусса.
5. Разработать браузерное приложение для интерполяции значений функции по заданным точкам методом наименьших квадратов.
6. Разработать браузерное приложение для кластеризации введенных двумерных точек методом k-means.
7. Разработать браузерную игру «Космические захватчики»
8. Разработать браузерное приложение для подсчета изолированных комнат в заданном лабиринте.

## КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ

1. Перечислите типы данных в JS.
2. Опишите назначение и приведите синтаксис оператора typeof.
3. Приведите особенности рператора typeof.
4. Сравните использование ключевых слов var и let.
5. Опишите механизм прототипного наследования.
6. Приведите синтаксис создания функции.
7. Охарактеризуйте контекст в js.
8. Опишите механизм самовызываемых функций.

9. Охарактерезуйте DOM.
10. Перечислите и охарактеризуйте методы для поиска элементов в DOM.
11. Перечислите действия, которые можно совершать с элементами DOM.
12. Приведите пример обработки событий в JS.
13. Опишите процедуру всплытия и погружения события
14. Приведите пример асинхронной загрузки JS-скриптов.
15. Опишите механизм работы промисов.
16. Приведите пример создания цепочки промисов.
17. Приведите пример рисования примитивов на canvas с помощью JS.
18. Опишите механизм создания анимации на canvas с помощью JS.

## ФОРМА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

На выполнение лабораторной работы отводится 7 часов: 6 часов на выполнение работы, 1 час на подготовку и защиту отчета по лабораторной работе.

Структура отчета: титульный лист, цель и задачи, формулировка задания (вариант), этапы выполнения работы, исходный код разработанного сайта, результаты выполнения работы (скриншоты), выводы.