Лабораторная работа №1
Основы JavaScript

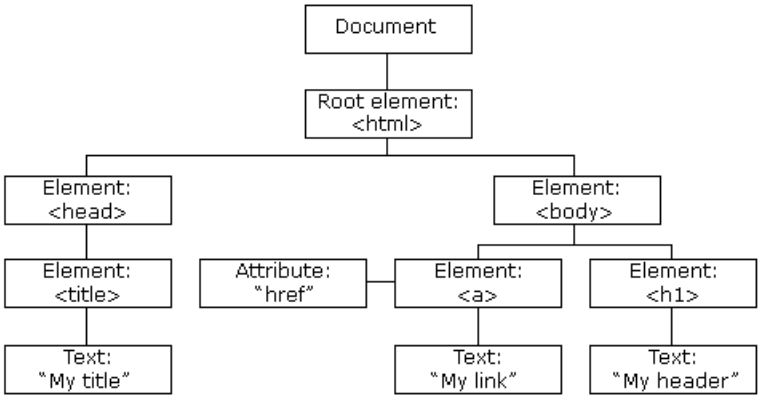
JavaScript был создан программистом **Brendan Eich**из Netscape и представлен в декабре 1995 года под названием LiveScript. Довольно быстро он был переименован в JavaScript, хотя официальным названием JavaScript является ECMAScript. ECMAScript разрабатывается и поддерживается Международной организацией ECMA (Европейская ассоциация производителей компьютеров).

Что такое JavaScript?

1) JavaScript — язык сценариев, или скриптов. Скрипт представляет собой программный код — набор инструкций, который не требует предварительной обработки (например, компиляции) перед запуском. Код JavaScript интерпретируется движком браузера во время загрузки веб-страницы. Интерпретатор браузера выполняет построчный анализ, обработку и выполнение исходной программы или запроса.

2) JavaScript — объектно-ориентированный язык с прототипным наследованием. Он поддерживает несколько встроенных объектов, а также позволяет создавать или удалять свои собственные (пользовательские) объекты. Объекты могут наследовать свойства непосредственно друг от друга, образуя цепочку объект-прототип.

1. Структура DOM (Document Object Model) документа HTML и место Javascript в теле документа



С помощью программируемой объектной модели JavaScript становится полноценным инструментом по созданию динамического HTML (DHTML):

- JavaScript может изменить все HTML элементы на странице
- JavaScript может изменить все атрибуты HTML на странице
- JavaScript может изменить все стили CSS на странице
- JavaScript может реагировать на все события на странице

Скрипты могут располагаться как в области заголовка HTML, так и в области тела HTML.

Пример структуры HTML и места скрипта в теле документа (создать приведенный пример документа. Может быть использован Visual Studio Code или любой другой редактор. Сохранить файл с названием Пример1_1.html и открыть (запустить) его в любом браузере).

```
<!DOCTYPE html>
<html>
        <head>
        <!-- область заголовка HTML - это комментарий в стандарте HTML -->
        </head>
        <body>
```

```
        <!-- область тела документа HTML -->
                <script>
//пример встраивания javascript в тело документа
/*использован метод Writeдля вывода на страницу результата выполнения
функции Date() – возвращение текущего даты/времени
*/
document.write(Date());
                </script>
        </body>
</html>
```

Пример кода в составе страницы HTML (Пример1_2.html**):**

```
<!DOCTYPE html>
        <html>
                <body>
                <p>
                JavaScript может написать прямо в HTML выходной поток – в теле документа
                </p>
<script>
        document.write("<h1>Это тег для заголовка</h1>");
        document.write("<p>Это тег для обозначения параграфа</p>");
</script>
                <p>
Вы можете использовать метод <strong> document.write </strong> в теле выходном
HTML.
Если вы используете этот метод после загрузки документа (например, в функции), весь
документ будет перезаписан.
<!--тег <strong> делает выделения текста на выходе страницы -->
                </p>
                </body>
    </html>
```

2. Подключение сценариев к html-документу

Сценарии JavaScript бывают встроенные, т.е. их содержимое является частью документа, и внешние, хранящиеся в отдельном файле с расширением .js. Сценарии можно внедрить в html-документ следующими способами:

1.1. В виде гиперссылки.

Для этого нужно разместить код в отдельном файле и включить ссылку на файл в заголовок

```
1    <head>
2    <script src="script.js"></script>
3    </head>
```

или тело страницы.

```
1    <body>
2    <script src="script.js"></script>
3    </body>
```

Этот способ обычно применяется для сценариев большого размера или сценариев, многократно используемых на разных веб-страницах.

1.2. Внутрь элемента <script>.

Элемент <script> может вставляться в любое место документа. Внутри тега располагается код, кото-

рый выполняется сразу после прочтения браузером, или содержит описание функции, которая выполняется в момент ее вызова. Описание функции можно располагать в любом месте, главное, чтобы к моменту ее вызова код функции уже был загружен.

Обычно код JavaScript размещается в заголовке документа (элемент <head>) или после открывающего тега <body>. Если скрипт используется после загрузки страницы, например, код счетчика, то его лучше разместить в конце документа:

```
1    <footer>
2    <script>
3    document.write("Введите свое имя");
4    </script>
5    </footer>
6    </body>
```

1.3. В виде обработчика события.

Каждый html-элемент имеет JavaScript-события, которые срабатывают в определенный момент. Нужно добавить необходимое событие в html-элемент как атрибут, а в качестве значения этого атрибута указать требуемую функцию. Функция, вызываемая в ответ на срабатывание события, является обработчиком события. В результате срабатывания события исполнится связанный с ним код. Этот способ применяется в основном для коротких сценариев, например, можно установить смену цвета фона при нажатии на кнопку:

html-код:

```
<button onclick="changeColor();">Сменить цвет фона</button>
```

css-код:

```
body { background: aliceblue }
```

js-код:

```
let colorArray = ["#5A9C6E", "#A8BF5A", "#FAC46E", "#FAD5BB",
"#F2FEFF"];
let i = 0;

function changeColor(){
    document.body.style.background = colorArray[i];
    i++;
    if( i > colorArray.length - 1){
        i = 0;
    }
}
```

2. Типы данных и переменные в JavaScript

Компьютеры обрабатывают информацию — данные. Данные могут быть представлены в различных формах или типах. Большая часть функциональности JavaScript реализуется за счет простого набора объектов и типов данных. Функциональные возможности, связанные со строками, числами и логикой, базируются на строковых, числовых и логических типах данных. Другая функциональная возможность, включающая регулярные выражения, даты и математические операции, осуществляется с помощью объектов RegExp, Date и Math.

Литералы в JavaScript представляют собой особый класс типа данных, фиксированные значения одного из трех типов данных — строкового, числового или логического:

```
1    "это строка"
2    3.14
3    true
```

```
1    alert("Hellow"); // "Hellow" - это литерал
2    let myLetiable = 15; // 15 - это литерал
```

Примитивный тип данных является экземпляром определенного типа данных, таких как строковый, числовой, логический, null и undefined.

2.1. Переменные в JavaScript

Данные, обрабатываемые сценарием JavaScript, являются переменными. Переменные представляют собой именованные контейнеры, хранящие данные (значения) в памяти компьютера, которые могут изменяться в процессе выполнения программы. Переменные имеют имя, тип и значение.

Имя переменной, или идентификатор, может включать только буквы a-z, A-Z, цифры 0-9 (цифра не может быть первой в имени переменной), символ $ (может быть только первым символом в имени переменной или функции) и символ подчеркивания _, наличие пробелов не допускается. Длина имени переменной не ограничена. Можно, но не рекомендуется записывать имена переменных буквами русского алфавита, для этого они должны быть записаны в Unicode.

В качестве имени переменной нельзя использовать ключевые слова JavaScript. Имена переменных в JavaScript чувствительные к регистру, что означает, что переменная let message; и let Message; — разные переменные.

Переменная создается (объявляется) с помощью ключевого слова let, за которым следует имя переменной, например, let message;. Объявлять переменную необходимо перед ее использованием.

Переменная инициализируется значением с помощью операции присваивания =, например, let message="Hellow";, т.е. создается переменная message и в ней сохраняется ее первоначальное значение "Hellow". Переменную можно объявлять без значения, в этом случае ей присваивается значение по умолчанию undefined. Значение переменной может изменяться во время исполнения скрипта. Разные переменные можно объявлять в одной строке, разделив их запятой:

let message="Hellow", number_msg = 6, time_msg = 50;

2.2. Типы данных переменных

JavaScript является нетипизированным языком, тип данных для конкретной переменной при ее объявлении указывать не нужно. Тип данных переменной зависит от значений, которые она принимает. Тип переменной может изменяться в процессе совершения операций с данными (динамическое приведение типов). Преобразование типов выполняется автоматически в зависимости от того, в каком контексте они используются. Например, в выражениях, включающих числовые и строковые значения с оператором +, JavaScript преобразует числовые значения в строковые:

```
let message = 10 + " дней до отпуска";
// вернет "10 дней до отпуска"
```

Получить тип данных, который имеет переменная, можно с помощью операторatypeof. Этот оператор возвращает строку, которая идентифицирует соответствующий тип.

```
typeof 35; // вернет "number"
typeof "text"; // вернет "string"
typeof true; // вернет "boolean"
typeof [1, 2, 4]; // вернет "object"
typeof undefined; // вернет "undefined"
typeof null; // вернет "object"
```

Все типы данных в JavaScript делятся на две группы — простые типы данных*(primitive data types)* и составные типы данных *(composite data types).*

К простым типам данных относят строковый, числовой, логический, null иunderfined.

2.2.1. Строковый тип (string)

Используется для хранения строки символов, заключенных в двойные или одинарные кавычки. Пустой набор символов, заключенный в одинарные или двойные кавычки, является пустой строкой. Число, заключенное в кавычки, также является строкой.

```
let money = ""; // пустая строка, ноль символов
let work = 'test';
let day = "Sunday";
let x = "150";
```

В строку в двойных кавычках можно включить одиночную кавычку и наоборот. Кавычка того же типа отключается с помощью символа обратного слэша \ (так называемая escape-последовательность):

```
1  document.writeln("\"Доброе утро, Иван Иваныч!\"\n");
2  // выведет на экран "Доброе утро, Иван Иваныч!"
```

Строки можно сравнивать, а также объединять с помощью операции конкатенации+. Благодаря автоматическому приведению типов можно объединять числа и строки. Строки являются постоянными, после того, как строка создана, она не может быть изменена, но может быть создана новая строка путем объединения других строк.

2.2.2. Числовой тип (number)

Используется для числовых значений. Числа в языке JavaScript бывают двух типов: целые числа *(integer)* и числа с плавающей точкой *(floating-point number)*. Целочисленные величины могут быть положительными, например 1, 2, и отрицательными, например –1, –2, или равными нулю. 1 и 1.0 — одно и то же значение. Большинство чисел в JavaScript записываются в десятичной системе счисления, также может использоваться восьмеричная и шестнадцатеричная системы.

В десятичной системе значения числовых переменных задаются с использованием арабских цифр 1, 2, 3, 4, 5, 6, 7, 8, 9, 0.

В восьмеричном формате числа представляет собой последовательность, содержащая цифры от 0 до 7 и начинающаяся с префикса 0.

Для шестнадцатеричного формата добавляется префикс 0x (0X), за которым следует последовательность из цифр от 0 до 9 или букв от a (A) до f (F), соответствующие значениям от 10 до 15.

```
1  let a = 120; // целое десятичное числовое значение
2  let b = 012; // восьмеричный формат
3  let c = 0xfff; // шестнадцатеричный формат
4  let d = 0xACFE12; // шестнадцатеричный формат
```

Числа с плавающей точкой представляют собой числа с дробной десятичной частью, либо это числа, выраженные в экспоненциальном виде. Экспоненциальная запись чисел предполагает следующий вид: число с дробной десятичной частью, за ним следует буква e, которая может быть указана как в верхнем, так и в нижнем регистре, далее — необязательный знак + или - и целая экспонента.

```
1  let a = 6.24; // вещественное число
2  let b = 1.234E+2; // вещественное число, эквивалентно 1.234
3  let c = 6.1e-2; // вещественное число, эквивалентно 6.1 X 10⁻²
```

X 10²

2.2.3. Логический тип (boolean)

Данный тип имеет два значения, true (истина), false (ложь). Используется для сравнения и проверки условий.

```
1  let answer = confirm("Вам понравилась эта статья?\n
2  Нажмите ОК. Если нет, то нажмите Cancel.");
3  if (answer == true)
4  {
5  alert("Спасибо!");
   }
```

Также существуют специальные типы простых значений:

нулевой тип — данный тип имеет одно значение null, которое используется для представления несуществующих объектов.

неопределенный тип — тип переменной underfined означает отсутствие первоначального значения переменной, а также несуществующее свойство объекта.

Составные типы данных состоят из более чем одного значения. К ним относятся объекты и особые типы объектов — массивы и функции. Объекты содержат свойства и методы, массивы представляют собой индексированный набор элементов, а функции состоят из коллекции инструкций.

2.3. Глобальные и локальные переменные

Переменные по области видимости делятся на **глобальные** и **локальные**. Область видимости представляет собой часть сценария, в пределах которой имя переменной связано с этой переменной и возвращает ее значение. Переменные, объявленные внутри тела функции, называются **локальными**, их можно использовать только в этой функции. Локальные переменные создаются и уничтожаются вместе с соответствующей функцией.

Переменные, объявленные внутри элемента <script>, или внутри функции, но без использования ключевого слова let, называются **глобальными**. Доступ к ним может осуществляться на протяжении всего времени, пока страница загружена в браузере. Такие переменные могут использоваться всеми функциями, позволяя им обмениваться данными.

Глобальные переменные попадают в **глобальное пространство имен**, которое является местом взаимодействия отдельных компонентов программы. Не рекомендуется объявлять переменные таким способом, так как аналогичные имена переменных уже могут использоваться любым другим кодом, вызывая сбой в работе скрипта.

Глобальное пространство в JavaScript представляется глобальным объектом window. Добавление или изменение глобальных переменных автоматически обновляет глобальный объект. В свою очередь, обновление глобального объекта автоматически приводит к обновлению глобального пространства имен.

Если глобальная и локальная переменная имеют одинаковые имена, то локальная переменная будет иметь преимущество перед глобальной.

Локальные переменные, объявленные внутри функции в разных блоках кода, имеют одинаковые области видимости. Тем не менее, рекомендуется помещать объявления всех переменных в начале функции.

Лабораторная работа № 2
Основы JavaScript. Структура кода
Написание команд

1. Создайте HTML-документ и подключим его в виде гиперссылки (Пример1_3.html).

```
<!DOCTYPE html>

<head>
<meta charset="windows-1251" />
<title>JavaScript Пример 1</title>
<script src="script.js"></script>
</head>

<body>

</body>
</html>
```

2. Для рассмотрения возможностей JavaScript создадим файл `script.js`

3. **ВЗАИМОДЕЙСТВИЕ С ПОЛЬЗОВАТЕЛЕМ: `ALERT, PROMPT, CONFIRM`**

При использовании функций ALERT, PROMPT, CONFIRM выводится модальное окно с вопросом – обычно это центр браузера, и внешний вид окна выбирает браузер. Разработчик не может на это влиять. С одной стороны – это недостаток, так как нельзя вывести окно в своем, особо красивом, дизайне. С другой стороны, преимущество этих функций по сравнению с другими, более сложными методами взаимодействия, которые мы изучим в дальнейшем – как раз в том, что они очень просты.

Это самый простой способ вывести сообщение или получить информацию от посетителя. Поэтому их используют в тех случаях, когда простота важна, а всякие «красивости» особой роли не играют.

`alert`

Синтаксис:

`alert(сообщение)`

alert выводит на экран окно с сообщением и приостанавливает выполнение скрипта, пока пользователь не нажмёт «ОК».

```
alert( "Привет" );
```

Окно сообщения, которое выводится, является модальным окном. Слово «модальное» означает, что посетитель не может взаимодействовать со страницей, нажимать другие кнопки и т.п., пока не разберётся с окном. В данном случае – пока не нажмёт на «ОК».

Можно вместо одного вызова `alert` сделать два (введите ниже приведенный текст в файл `script.js` и просмотрите результат):

```
alert('Привет'); alert('Мир');
```

или

```
alert('Привет');
alert('Мир');
```

Точку с запятой во многих случаях можно не ставить, если есть переход на новую строку.

```
alert('Привет')
alert('Мир')
```

В этом случае JavaScript интерпретирует переход на новую строчку как разделитель команд и автоматически вставляет «виртуальную» точку с запятой между ними. Однако, внутренние правила по вставке точки с запятой не идеальны. В примере выше они сработали, но в некоторых ситуациях JavaScript «забывает» вставить точку с запятой там, где она нужна. Таких ситуаций не так много, но они все же есть, и ошибки, которые при этом появляются, достаточно сложно исправлять.

Поэтому рекомендуется точки с запятой ставить. Сейчас это, фактически, стандарт.

`prompt`

Функция prompt принимает два аргумента:

```
result = prompt(title, default);
```

Она выводит модальное окно с заголовком title, полем для ввода текста, заполненным строкой по умолчанию default и кнопками OK/CANCEL. Пользователь должен либо что-то ввести и нажать OK, либо отменить ввод кликом на CANCEL или нажатием Esc на клавиатуре. Вызов prompt возвращает то, что ввёл посетитель – строку или специальное значение null, если ввод отменён.

Например (вставьте код в `script.js`):

```
let years = prompt('Сколько вам лет?', 100);

alert('Вам ' + years + ' лет!')
```

`confirm`

Синтаксис:

result = confirm(question);

confirm выводит окно с вопросом question с двумя кнопками: OK и CANCEL.

Результатом будет true **при нажатии OK и** false **– при CANCEL(**Esc**).**

Например (вставьте код в `script.js`):

```
let isAdmin = confirm("Вы - администратор?");

alert( isAdmin );
```

4. Комментарии могут находиться в любом месте программы и никак не влияют на ее выполнение. Интерпретатор JavaScript попросту игнорирует их.

Однострочные комментарии начинаются с двойного слэша //

alert('Мир'); // Второе сообщение выводим отдельно

Многострочные комментарии начинаются слешем-звездочкой "/*" и заканчиваются звездочкой-слэшем "*/".
Вложенные комментарии не поддерживаются!

5. Рассмотрим пример структуры HTML и места скрипта в теле документа (Primer_2.html)
При добавлении скрипта в тело документа воспользуемся методом **document.write**
Метод **document.write** – один из наиболее древних методов добавления текста к документу.… Как работает document.write. Метод document.write(str) работает только пока HTML-страница находится в процессе загрузки. Нет никаких ограничений на содержимое document.write. Метод document.write очень быстро (Пример1_4.html).
.

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<script>
//пример встраивания javascript в тело документа
/*использован метод Write для вывода на страницу результата выполнения
функции Date() – возвращение текущего даты/времени
*/
document.write(Date());
</script>
</body>
</html>
```

6. Рассмотрим пример структуры HTML и места скрипта в теле документа (Пример1_5.html).
:

```
<!DOCTYPE html>
<html>
<body>
<p>
JavaScript может написать прямо в HTML выходной поток – в теле документа
</p>
<script>
document.write("<h1>Это тег для заголовка</h1>");
document.write("<p>Это тег для обозначения параграфа</p>");
</script>
<p>
Вы можете использовать метод <strong> document.write </strong> в теле выходном
HTML.
Если вы используете этот метод после загрузки документа (например, в функции), весь
документ будет перезаписан.
<!--тег <strong> делает выделения текста на выходе страницы -->
</p>
</body>
</html>
```

7. Использование переменных
Для объявления или, другими словами, создания переменной используется ключевое слово

let:

let message;

После объявления, можно записать в переменную данные:

let message;

message = 'Привет'; // сохраним в переменной строку

Эти данные будут сохранены в соответствующей области памяти и в дальнейшем доступны при обращении по имени (Пример1_6.html).

):

```
<!DOCTYPE html>
<html>
<body>
<p>
    Использование переменных
</p>
<script>
    let message;
    message = 'Привет';
    alert(message); // выведет содержимое переменной
</script>
</body>
</html>
```

Для краткости можно совместить объявление переменной и запись данных:

```
let message = 'Привет';
```

Переменные в JavaScript могут хранить не только строки, но и другие данные, например, числа.

Объявим две переменные, положим в одну - строку, а в другую - число.

Как вы можете видеть, переменной без разницы, что хранить:

```
let num = 100500;
let message = 'Привет';
```

Значение можно копировать из одной переменной в другую.

```
let num = 100500;
let message = 'Привет';
message = num;
```

Значение из num перезаписывает текущее в message.

В JavaScript вы можете создать переменную и без let, достаточно просто присвоить ей значение:

```
x = "value"; // переменная создана, если ее не было
```

Технически, это не вызовет ошибки, но делать так все-таки не стоит.

Всегда определяйте переменные через let. Это хороший тон в программировании и помогает избежать ошибок.

Пример документа с объявлением переменных:

```
<html>
<body>
<div id="test"></div>
<script>
let test = 5;
alert(test);
</script>
</body>
</html>
```

8. Задание: Создайте документ HTML, в котором средствами JavaScript:
- Объявите две переменные: admin и name.
- Запишите в name строку "Ваше имя".
- Скопируйте значение из name в admin.
- Выведите admin (должно вывести «Ваше имя»).

9. Объявление констант

Константа — это переменная, которая никогда не меняется. Как правило, их называют большими буквами, через подчёркивание.

```
let COLOR_BLUE = "#00F";
let COLOR_RED = "#0F0";
let COLOR_GREEN = "#F00";
let COLOR_ORANGE = "#FF7F00";
alert(COLOR_RED); // #0F0
```

Технически, константа является обычной переменной, то есть её можно изменить. Но мы договариваемся этого не делать.

Зачем нужны константы? Почему бы просто не использовать "#F00" или "#0F0"?

1. Во-первых, константа — это понятное имя, в отличие от строки "#FF7F00".
2. Во-вторых, опечатка в строке может быть не замечена, а в имени константы её упустить невозможно — будет ошибка при выполнении.

Константы используют вместо строк и цифр, чтобы сделать программу понятнее и избежать ошибок.

На имя переменной наложены два ограничения:

1. Имя может состоять из: букв, цифр, символов $ и _
2. Первый символ не должен быть цифрой.

!!! Регистр букв имеет значение

Переменные apple и AppLE - две разные переменные.

Существует список зарезервированных слов, которые нельзя использовать при именовании переменных, так как они используются самим языком, например: let, class, return, implements и др.

Некоторые слова, например, class, не используются в современном JavaScript, но они заняты на будущее. Некоторые браузеры позволяют их использовать, но это может привести к ошибкам.

9. **Типы данных в JavaScript**

Число **number**:

```
let n = 123;
n = 12.345;
```

Строка **string**:

```
let str = "Мама мыла раму";
str = 'Одинарные кавычки тоже подойдут';
```

1. В JavaScript одинарные и двойные кавычки равноправны. Можно использовать или те или другие.
2. Тип символ не существует, есть только строка
3. В некоторых языках программирования есть специальный тип данных для одного символа. Например, в языке C это char. В JavaScript есть только тип «строка» string. Что, надо сказать, вполне удобно.

Булевый (логический) тип **boolean**. У него всего два значения - true (истина) и false (ложь).

Как правило, такой тип используется для хранения значения типа да/нет, например:

```
let checked = true; // поле формы помечено галочкой
checked = false; // поле формы не содержит галочки
```

Мы поговорим более подробно, когда будем обсуждать логические вычисления и условные операторы.

null — специальное значение. Оно имеет смысл «ничего». Значение null не относится ни к одному из типов выше, а образует свой отдельный тип, состоящий из единственного значения null:

let age = null;

1. В JavaScript null не является «ссылкой на несуществующий объект» или «нулевым указателем», как в некоторых других языках. Это просто специальное значение, которое имеет смысл «ничего» или «значение неизвестно».

2. В частности, код выше говорит о том, что возраст age неизвестен.

3. **undefined** — специальное значение, которое, как и null, образует свой собственный тип. Оно имеет смысл «значение не присвоено».

Если переменная объявлена, но в неё ничего не записано, то ее значение как раз и есть undefined:

let u;

alert(u); // выведет "undefined"

Можно присвоить undefined и в явном виде, хотя это делается редко:

let x = 123;

x = undefined;

В явном виде undefined обычно не присваивают, так как это противоречит его смыслу. Для записи в переменную «пустого значения» используется null.

Объекты object

Первые 5 типов называют «примитивными».

Особняком стоит шестой тип: «объекты». К нему относятся, например, даты, он используется для коллекций данных и для многого другого.

ИТОГО: Есть 5 «примитивных» типов: number, string, boolean, null, undefined и объекты object.

10. Операторы

Для работы с переменными, со значениями, JavaScript поддерживает все стандартные операторы, большинство которых есть и в других языках программирования.

1. Термины: «унарный», «бинарный», «операнд»

У операторов есть своя терминология, которая используется во всех языках программирования.

 Операнд — то, к чему применяется оператор. Например: 5 * 2 — оператор умножения с левым и правым операндами. Другое название: «аргумент оператора».

 Унарным называется оператор, который применяется к одному выражению.

Например, оператор унарный минус "-" меняет знак числа на противоположный:

```
var x = 1;
alert( -x ); // -1, унарный минус
alert( -(x+2) ); // -3, унарный минус применён к результату сложения x+2
alert( -(-3) ); // 3
```

 Бинарным называется оператор, который применяется к двум операндам. Тот же минус существует и в бинарной форме:

```
var x = 1, y = 3;
alert( y - x ); // 2, бинарный минус
```

Работа унарного "+" и бинарного "+" в JavaScript существенно различается.

Это действительно разные операторы. Бинарный плюс складывает операнды, а унарный — ничего не делает в арифметическом плане, но зато приводит операнд к числовому типу.

10.1. Арифметические операторы

Арифметические операторы предназначены для выполнения математических операций, они работают с числовыми операндами (или переменными, хранящими числовые значения), возвращая в качестве результата числовое значение.

Если один из операндов является строкой, интерпретатор JavaScript попытается преобразовать его в числовой тип, а после выполнить соответствующую операцию. Если преобразование типов окажется невозможным, будет получен результат NaN(не число).

```
let x = 5, y = 8, z;
z = x + y; // вернет 13
z = x - y; // вернет -3
z = - y; // вернет -8
z = x * y; // вернет 40
z = x / y; // вернет 0.625
```

z = y % x; // вернет 3. Вычисляет остаток, получаемый при целочисленном делении первого операнда на второй. Применяется как к целым числам, так и числам с плавающей точкой

10.2. Сложение строк, бинарный +

Если бинарный оператор + применить к строкам, то он их объединяет в одну:

```
var a = "моя" + "строка";
alert(a); // моястрока
```

Если хотя бы один аргумент является строкой, то второй будет также преобразован к строке!

Причем не важно, справа или слева находится операнд-строка, в любом случае нестроковый аргумент будет преобразован. Например:

```
alert( '1' + 2 ); // "12"
alert( 2 + '1' ); // "21"
```

Это приведение к строке — особенность бинарного оператора "+".

Остальные арифметические операторы работают только с числами и всегда приводят аргументы к числу.

Например:

```
alert( '1' - 2 ); // -1
alert( 6 / '2'); // 3
```

10.3. Унарный плюс +

Унарный плюс как арифметический оператор ничего не делает:

```
alert( +1 ); // 1
alert( +(1-2) ); // -1
```

Как видно, плюс ничего не изменил в выражениях. Результат — такой же, как и без него.

Тем не менее, он широко применяется, так как его «побочный эффект» — преобразование значения в число.

Например, у нас есть два числа, в форме строк, и нужно их сложить. Бинарный плюс сложит их как строки, поэтому используем унарный плюс, чтобы преобразовать к числу:

```
var a = "2";
var b = "3";
alert( a + b ); // 23, так как бинарный плюс складывает строки
alert( +a + b ); // 23, второй операнд - всё ещё строка
alert( +a + +b); // 5, оба операнда предварительно преобразованы в числа
```

11. Операторы присваивания

Операторы присваивания используются для присваивания значений переменным. Комбинированные операторы позволяют сохранить первоначальное и последующее значение в одной переменной.

Оператор/Операция	Описание
= Присваивание	Используется для присваивания значения переменной.

+=, −=, *=, /=,%=Комбинированный оператор	Выполняет присваивание с операцией. Между первым и вторым операндом выполняется соответствующая операция, затем результат присваивается первому операнду.

```
let a = 5; // присваиваем переменной a числовое значение 5
let b = "hellow"; // сохраняем в переменной b строку hellow
let m = n = z = 10; // присваиваем переменным m, n, z числовое значение 10
x += 10; // равнозначно x = x + 10;
x -= 10; // равнозначно x = x - 10;
x *= 10; // равнозначно x = x * 10;
x /= 10; // равнозначно x = x / 10;
x %= 10; // равнозначно x = x % 10;
```

12. Приоритет

В том случае, если в выражении есть несколько операторов - порядок их выполнения определяется приоритетом.

Из школы мы знаем, что умножение в выражении 2 * 2 + 1 выполнится раньше сложения, т.к. его приоритет выше, а скобки явно задают порядок выполнения. Но в JavaScript — гораздо больше операторов, поэтому существует целая таблица приоритетов.

Она содержит как уже пройденные операторы, так и те, которые мы еще не проходили. В ней каждому оператору задан числовой приоритет. Тот, у кого число меньше — выполнится раньше. Если приоритет одинаковый, то порядок выполнения — слева направо.

Отрывок из таблицы:

… … …
5 умножение *
5 деление /
6 сложение +
6 вычитание -
17 присвоение =
… … …

Посмотрим на таблицу в действии.

В выражении x = 2 * 2 + 1 приоритет умножения * равен 5, он самый высокий, поэтому выполнится раньше всех. Затем произойдёт сложение +, у которого приоритет 6, и после них — присвоение =, с приоритетом 17.

13. Операторы инкремента и декремента

Операции инкремента и декремента являются унарными и производят увеличение и уменьшение значения операнда на единицу. В качестве операнда может быть переменная, элемент массива, свойство объекта. Чаще всего такие операции используются для увеличения счетчика в цикле.

Оператор/Операция	Описание
++x Префиксный инкремент	Увеличивает операнд на единицу.
x++ Постфиксный инкремент	Прибавляет к операнду единицу, но результатом выражения будет являться первоначальное значение операнда.
−−x Префиксный декремент	Уменьшает на единицу операнд, возвращая уменьшенное значение.
x− Постфиксный декремент	Уменьшает на единицу операнд, возвращая первоначальное значение.

Итак:

 Инкремент ++ увеличивает на 1:

```
var i = 2;
i++; // более короткая запись для i = i + 1.
alert(i); // 3
```

 Декремент -- уменьшает на 1:

```
var i = 2;
i--; // более короткая запись для i = i - 1.
```

```
alert(i); // 1
```
Инкремент/декремент можно применить только к переменной.
Код 5++ даст ошибку.

Например:

```
let x = y = m = n = 5, z, s, k, l;
z = ++x * 2; /* в результате вычислений вернет значение z = 12, x = 6, т.е. значение x сначала увеличивается на 1, а после выполняется операция умножения */
s = y++ * 2; /* в результате вычислений вернет значение s = 10, y = 6, т.е. сначала выполняется операция умножения, а после в переменной y сохраняется увеличенное на 1 значение */
k = --m * 2; // вернет значение k = 8, m = 4
l = n-- * 2; // вернет значение l = 10, n = 4
```

Вызывать эти операторы можно не только после, но и перед переменной: i++ (называется «постфиксная форма») или ++i («префиксная форма»).
Обе эти формы записи делают одно и то же: увеличивают на 1.
Тем не менее, между ними существует разница. Она видна только в том случае, когда мы хотим не только увеличить/уменьшить переменную, но и использовать результат в том же выражении.
Например:

```
var i = 1;
var a = ++i; // (*)
alert(a); // 2
```

В строке (*) вызов ++i увеличит переменную, а затем вернёт ее значение в a. **То есть, в a попадёт значение i после увеличения.**
Постфиксная форма i++ отличается от префиксной ++i тем, что возвращает старое значение, бывшее до увеличения.
В примере ниже в a попадёт старое значение i, равное 1:

```
var i = 1;
var a = i++; // (*)
alert(a); // 1
```

- Если результат оператора не используется, а нужно только увеличить/уменьшить переменную — без разницы, какую форму использовать::

```
var i = 0;
i++;
++i;
alert(i); // 2
```

- Если хочется тут же использовать результат, то нужна префиксная форма:

```
var i = 0;
alert( ++i ); // 1
```

- Если нужно увеличить, но нужно значение переменной до увеличения — постфиксная форма:

```
var i = 0;
alert( i++ ); // 0
```

Инкремент/декремент можно использовать в любых выражениях.
При этом он имеет более высокий приоритет и выполняется раньше, чем арифметические операции:

```
var i = 1;
alert( 2 * ++i ); // 4

var i = 1;
alert( 2 * i++ ); // 2, выполнился раньше но значение вернул старое
```

При этом, нужно с осторожностью использовать такую запись, потому что при чтении кода зачастую неочевидно, что переменая увеличивается. Три строки — длиннее, зато нагляднее:

```
var i = 1;

var i = 1;
alert( 2 * i );
i++;
```

14. Побитовые операторы

Побитовые операторы рассматривают аргументы как 32-разрядные целые числа и работают на уровне их внутреннего двоичного представления.

Эти операторы не являются чем-то специфичным для JavaScript, они поддерживаются в большинстве языков программирования.

Поддерживаются следующие побитовые операторы:

- AND(и) (&)
- OR(или) (|)
- XOR(побитовое исключающее или) (^)
- NOT(не) (~)
- LEFT SHIFT(левый сдвиг) (<<)
- RIGHT SHIFT(правый сдвиг) (>>)
- ZERO-FILL RIGHT SHIFT(правый сдвиг с заполнением нулями) (>>>)

15. Вызов операторов с присваиванием

Часто нужно применить оператор к переменной и сохранить результат в ней же, например:

```
n = n + 5;
d = d * 2;
```

Эту запись можно укоротить при помощи совмещённых операторов: +=, -=, *=, /=, >>=, <<=, >>>=, &=, |=, ^=, вот так:

```
var n = 2;
n += 5; // теперь n=7 (работает как n = n + 5)
n *= 2; // теперь n=14 (работает как n = n * 2)
alert(n); // 14
```

Все эти операторы имеют в точности такой же приоритет, как обычное присваивание, то есть выполняются после большинства других операций.

Важность: 3

Чему будет равен x в примере ниже?

```
var a = 2;
```

16. Оператор запятая

Запятая тоже является оператором. Ее можно вызвать явным образом, например:

```
a = (5, 6);
alert(a);
```

Запятая позволяет перечислять выражения, разделяя их запятой ','. Каждое из них — вычисляется и отбрасывается, за исключением последнего, которое возвращается.

Запятая — единственный оператор, приоритет которого ниже присваивания. В выражении a = (5,6) для явного задания приоритета использованы скобки, иначе оператор '=' выполнился бы до запятой ',', получилось бы (a=5), 6.

Зачем же нужен такой странный оператор, который отбрасывает значения всех перечисленных выражений, кроме последнего?

Обычно он используется в составе более сложных конструкций, чтобы сделать несколько действий в одной строке. Например:

```
// три операции в одной строке
for (a = 1, b = 3, c = a*b; a < 10; a++) {
...
}
```

Такие трюки используются во многих JavaScript-фреймворках для укорачивания кода.
Пример с массивом и циклом (Пример1_7.html).
:

```
<!DOCTYPE html>
<html>
<body>
<script>
var i;
var cars = new Array();
cars[0] = "Saab";
cars[1] = "Volvo";
cars[2] = "BMW";
for (i=0;i<cars.length;i++)
{
document.write(cars[i] + "<br>");
}
</script>
</body>
</html>
```

Сохранить файл с названием Пример1_7.html и открыть (запустить) его в любом браузере.

13. Операторы сравнения

Операторы сравнения используются для сопоставления операндов, результатом выражения может быть одно из двух значений — true или false. Операндами могут быть не только числа, но и строки, логические значения и объекты. Однако сравнение может выполняться только для чисел и строк, поэтому операнды, не являющиеся числами или строками, преобразуются.

Если оба операнда не могут быть успешно преобразованы в числа или строки, операторы всегда возвращают false.

Если оба операнда являются строками/числами или могут быть преобразованы в строки/числа, они будут сравниваться как строки/числа.

Если один операнд является строкой/преобразуется в строку, а другой является числом/преобразуется в число, то оператор попытается преобразовать строку в число и выполнить сравнение чисел. Если строка не является числом, она преобразуется в значение NaN и результатом сравнения будет false.

Чаще всего операции сравнения используются при организации ветвлений в программах.

Оператор/Операция	Описание
== Равенство	Проверяет две величины на совпадение, допуская преобразование типов. Возвращает `true`, если операнды совпадают, и `false`, если они различны.
!=Неравенство	Возвращает `true`, если операнды не равны
===Идентичность	Проверяет два операнда на «идентичность», руководствуясь строгим определением совпадения. Возвращает `true`, если операнды равны без преобразования типов.
!==Неидентичность	Выполняет проверку идентичности. Возвращает`true`, если операнды не равны без преобразования типов.
> Больше	Возвращает `true`, если первый операнд больше второго, в противном случае возвращает `false`.
>= Больше или равно	Возвращает `true`, если первый операнд не меньше второго, в противном случае возвращает `false`.
< Меньше	Возвращает `true`, если первый операнд меньше второго, в противном случае возвращает `false`.
<= Меньше или равно	Возвращает `true`, если первый операнд не больше второго, в противном случае возвращает `false`.

```
5 == "5"; // вернет true
```

```
5 != -5.0; // вернет true
5 === "5"; // вернет false
false === false; // вернет true
1 !== true; // вернет true
1 != true; // вернет false, так как true преобразуется в 1
3 > -3; // вернет true
3 >= "4"; // вернет false
```

14. Логические операторы

Логические операторы позволяют комбинировать условия, возвращающие логические величины. Чаще всего используются в условном выражении if.

Оператор/Операция	Описание
&& Логическое И	Возвращает `true`, только если оба операнда истинны. При выполнении операции сначала проверяется значение первого операнда. Если оно имеет значение `false`, то значение второго оператора не учитывается и результату выражения присваивается `false`.
|| Логическое ИЛИ	Возвращает `true`, если хотя бы один операнд истинен, т.е. проверяет истинность как минимум одного условия.
! Логическое НЕ	Изменяет значение оператора на обратное - с `true` на `false` и наоборот.

```
 (2 < 3) && (3===3); // вернет true, так как выражения в обеих скобках дают true
(x < 10 && x > 0); // вернет true, если значение x принадлежит промежутку от 0 до 10
!false; // вернет true
```

15. Строковые операторы

Существует несколько операторов, которые работают со строками особым образом.

Оператор/Операция	Описание
+Конкатенация	Оператор работает слева направо, выполняя объединение строк. Если первый операнд является строкой, последующие операнды будут преобразованы в строки и далее выполнится их объединение.
+=Конкатенация с присваиванием	Выполняется объединение двух строк и результат присваивается переменной.
>, <, >=, <=,== Сравнение	Строки сравниваются по алфавиту, буквы в верхнем регистре всегда меньше букв в нижнем регистре. Сравнение строк основывается на номерах символов, указанных в стандарте Unicode, где прописные буквы идут раньше, чем строчные.

```
"1" + "10"; // вернет "110"
"1" + 10; // вернет "110"
2 + 5 + " цветных карандашей"; // вернет "7 цветных карандашей"
"Цветных карандашей " + 2 + 5; // вернет "Цветных карандашей 25"
"1" > "10"; // вернет false
"10" <= 10; // вернет true
"СССР" == "ссср"; // вернет false
x = "micro"; x+= "soft"; // вернет "microsoft"
```

16 . Пример объявления переменных и их использования (Пример1_8.html).

```
<!DOCTYPE html>
<html>
<body>
<script>
let pi=3.14;
let name="Григорий Остер";
let answer=' Пример объявления переменных';
document.write(pi + "<br>");
document.write(name + "<br>");
document.write(answer + "<br>");
</script>
</body>
</html>
```

Здесь в скрипте
 - это тег HTML, который осуществляет переход на новую строку.

17. **Функции**.

Зачастую нам надо повторять одно и то же действие во многих частях программы. Например, красиво вывести сообщение необходимо при приветствии посетителя, при выходе посетителя с сайта, ещё где-нибудь. Чтобы не повторять один и тот же код во многих местах, придуманы функции. Функции являются основными «строительными блоками» программы.

Примеры встроенных функций вы уже видели – это alert(message), prompt(message, default) и confirm(question). Но можно создавать и свои.

Рассмотрим пример с использованием тега <button> создает на веб-странице кнопки. На подобной кнопке можно размещать любые элементы HTML, в том числе изображения. Используя стили можно определить вид кнопки путем изменения шрифта, цвета фона, размеров и других параметров.

Создать приведенный пример документа (Пример1_9.html):

```
<!DOCTYPE html>
<html>
<body>
<p>Нажмите кнопку для объявления переменной и вывода результата.</p>
<button onclick="myFunction()">КНОПКА</button> //Событие onclick возникает при нажатии на кнопке
<p id="demo">Текст, помеченный меткой demo</p>
<script>
//объявляем функцию, которую будет вызывать кнопка по методу onclick
function myFunction()
{
//объявляем переменную строковую
let carname="Volvo";
/*
document.getElementById - метод объекта document. Он возвращает ссылку на узел документа,
которую можно использовать для изменения свойств и обращения к методам узла.
*/
//метод getElementById, получающий данные из тега по метке demo
// Свойство innerHTML используют для добавления текста
// Более подробное пояснение приведено ниже.
document.getElementById("demo").innerHTML=carname;
}
</script>
</body>
</html>
```

document.getElementById или просто id

Если элементу назначен специальный атрибут id, то можно получить его прямо по переменной с именем из значения id. Например:

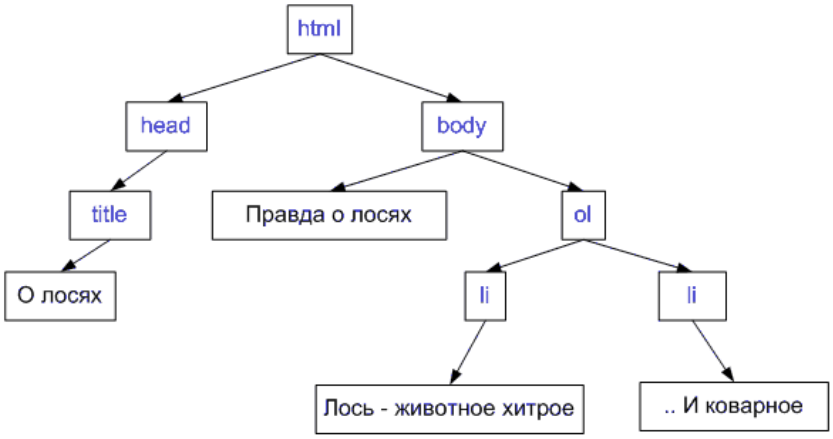
```
<div id="content-holder">
  <div id="content">Элемент</div>
</div>

<script>
  alert( content ); // DOM-элемент
  alert( window['content-holder'] );
</script>
```

P.S. ***DOM-элемент.*** *Согласно DOM-модели, документ является иерархией. Каждый HTML-тег образует отдельный элемент-узел, каждый фрагмент текста - текстовый элемент, и т.п. Проще говоря, DOM - это представление документа в виде дерева тегов. Это дерево образуется за счет вложенной структуры тегов плюс текстовые фрагменты страницы, каждый из которых образует отдельный узел.*

Рассмотрим дерево DOM для следующего документа. Самый внешний тег - <html>, поэтому дерево начинает расти от него. Теги образуют узлы-элементы (element node). Текст представлен текстовыми узлами(text node). И то и другое - равноправные узлы дерева DOM. На рисунке синим цветом обозначены элементы-узлы, черным - текстовые элементы. Дерево образовано за счет синих элементов-узлов - тегов HTML.

```
<html>
    <head>
        <title>
            О лосях
        </title>
    </head>
    <body>
        Правда о лосях.
        <ol>
            <li>
                Лось - животное хитрое
            </li>
            <li>
                .. И коварное
            </li>
        </ol>
    </body>
</html>
```



А вот так выглядит дерево, если изобразить его прямо на HTML-страничке. Кстати, дерево на этом рисунке не учитывает текст, состоящий из одних пробельных символов. Например, такой текстовый узел должен идти сразу после <ol>. DOM, не содержащий таких "пустых" узлов, называют "нормализованным".

Зачем нужна иерархическая модель DOM?

Каждый DOM-элемент является объектом и предоставляет свойства для манипуляции своим содержимым, для доступа к родителям и потомкам.

Для манипуляций с DOM используется объект document. Используя document, можно получать нужный элемент дерева и менять его содержание.

Итак, более правильным является доступ к элементу вызовом `document.getElementById("идентификатор")`.

Например:

```
div id="content">Выделим этот элемент</div>
<script>
  let elem = document.getElementById('content');
  elem.style.background = 'red';
  alert( elem == content ); // true
  content.style.background = ""; // один и тот же элемент
</script>
```

По стандарту значение `id` должно быть уникально, то есть в документе может быть только один элемент с данным `id`. И именно он будет возвращён. Если в документе есть несколько элементов с уникальным `id`, то поведение неопределено. То есть, нет гарантии, что браузер вернёт именно

первый или последний – вернёт случайным образом. Поэтому стараются следовать правилу уникальности `id`.

18. **Пример на Undefined and Null** (Пример1_10.html)**:**

```
<!DOCTYPE html>
<html>
<body>
<script>
let person;
let car="Volvo";
// document.write- метод, выводящий на страницу переданные ему аргументы
document.write(персона + "<br>");
document.write(авто + "<br>");
let car=null
document.write(авто + "<br>");
</script>
</body>
</html>
```

18. Пример создает объект с названием "person" и добавляет 4 свойства объекту (Пример1_11.html)**:**

```
<!DOCTYPE html>
<html>
<body>
<script>
let person=new Object();
person.firstname="Алексею";
person.lastname="Иванову";
person.age=50;
person.eyecolor="blue";
document.write(person.firstname + " исполнилось " + person.age + " лет.");</script>
</body>
</html>
```

19. Пример с массивом и циклом (Пример1_12.htmlинд
)**:**

```
<!DOCTYPE html>
<html>
<body>
<script>
let i;
let cars = new Array();
cars[0] = "Saab";
cars[1] = "Volvo";
cars[2] = "BMW";
for (i=0;i<cars.length;i++)
{
document.write(cars[i] + "<br>");
}
</script>
</body>
</html>
```

ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ ПО ВАРИАНТАМ

Варианты распределяются по первой букве Вашей фамилии:
А…Е – 1 вариант;
Ж…М – 2 вариант;
Н…У – 3 вариант;
Ф…Я – 4 вариант

Задание инд 1:

Вариант 1: Создать страницу со скриптом, который бы средствами скрипта выводил sin(x), где x – числовая переменная, которой присвоено некоторое значение на выбор в скрипте
Вариант 2: Создать страницу со скриптом, который бы средствами скрипта выводил cos(x), где x – числовая переменная, которой присвоено некоторое значение на выбор в скрипте
Вариант 3: Создать страницу со скриптом, который бы средствами скрипта выводил sqrt(x), где x – числовая переменная, которой присвоено некоторое значение на выбор в скрипте
Вариант 4: Создать страницу со скриптом, который бы средствами скрипта выводил abs(x), где x – числовая переменная, которой присвоено некоторое значение на выбор в скрипте

Задание инд 2:

Вариант 1: Создать страницу со скриптом, которая выводила бы сообщение « Нажмите кнопку для замены текста страницы». В скрипте опишите тег <button>, который будет при нажатии будет запускать функцию, созданную средствами javascript, которая заменит текст, размещенный в теге <h1> </h1> на текст «произошла замена», который бы извлекался из переменной str_.
Вариант 2: Создать страницу со скриптом, которая выводила бы сообщение « Нажмите кнопку для замены текста страницы» с кнопками «да» и «нет». В скрипте опишите тег <button>, который будет при нажатии будет запускать функцию, созданную средствами javascript, которая заменит текст, размещенный в теге   на результат сложения двух переменных, которые объявлены в функции.
Вариант 3: Создать страницу со скриптом, которая выводила бы сообщение «Нажмите кнопку для замены текста страницы». В скрипте опишите тег <button>, который будет при нажатии будет запускать функцию, созданную средствами javascript, которая заменит текст, размещенный в теге <a> на текст «произошла замена ссылки», который бы извлекался из переменной str_, хранящей ссылку на сайт.
Вариант 4: Создать страницу со скриптом, которая выводила бы сообщение « Нажмите кнопку для замены текста страницы». В скрипте опишите тег <button>, который будет при нажатии будет запускать функцию, созданную средствами javascript, которая заменит текст, размещенный в теге <h6> </h6> на текст текущую дату, которая бы извлекалась из переменной date_.

Задание инд 3:

Вариант 1: Создать страницу со скриптом, в котором создается объект машина с тремя свойствами: цвет, марка, модель. Вывести по нажатию кнопки все значения свойств на экран.
Вариант 2: Создать страницу со скриптом, в котором создается объект Холодильник с 4-мя свойствами: цвет, марка, модель, цена. Вывести по нажатию кнопки все значения свойств на экран.
Вариант 3: Создать страницу со скриптом, в котором создается объект Компьютер с 3-мя свойствами: модель, производитель, цена. Вывести по нажатию кнопки все значения свойств на экран.
Вариант 4: Создать страницу со скриптом, в котором создается объект Квартира с 4-мя свойствами: адрес, количество комнат, цена, ремонт. Вывести по нажатию кнопки все значения свойств на экран.

Отчет по лабораторной работе

В соответствии со структурой заготовки отчета и примером оформления оформить в отчете все задания, выполняемые в ходе лабораторной работы, а также индивидуальные задания по вариантам. Файл с отчетом называть по шаблону: Фамилия_лаб_раб_номер.

Отчет предоставляется в электронном виде вместе с файлами задания и размещается на GitHabe либо присылается на электронную почту для проверки. Также по результатам лабораторной работы на следующем за ней занятии проводится выборочный опрос по командам языка.