Лабораторная работа №1
Основы javaScript

Цель:

Структура кода

Написание команд:
Например, можно вместо одного вызова alert сделать два:

```
<script>
alert('Привет'); alert('Мир');
</script>
```

```
<script>
alert('Привет');
alert('Мир');
</script>
```

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

```
<script>
alert('Привет')
alert('Мир')
</script>
```

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.

Написание комментариев

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 //

```
<script>
alert('Мир'); // Второе сообщение выводим отдельно
</script>
```

Многострочные комментарии начинаются слешем-звездочкой "/*" и заканчиваются звездочкой-слэшем "*/".

Вложенные комментарии не поддерживаются!

В этом коде будет ошибка:

```
<script>
/*
alert('Привет'); /* вложенный комментарий ?!? */
*/
alert('Мир');
</script>
```

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

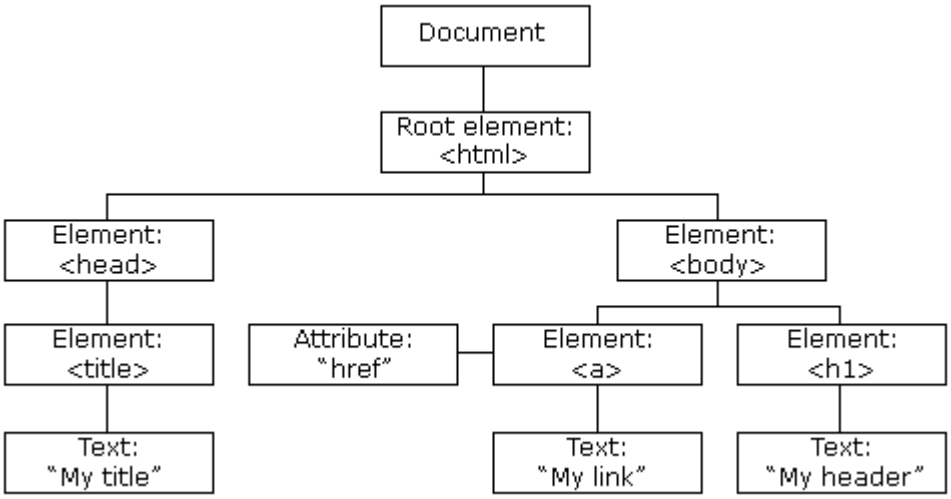

Рисунок - Структура DOM (Document Object Model) документа HTML

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:

```
<!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>
```

Задание: Создать приведенный пример документа в любом редакторе. Удобнее использовать редактор Visual Studio или любой другой, поддерживающий языки

разметки. Сохранить файл с названием Пример1_1.html и открыть (запустить) его в любом браузере.

Пример кода в составе страницы 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>
```

Задание: Создать приведенный пример документа в любом редакторе. Удобнее использовать редактор Visual Studio или любой другой, поддерживающий языки разметки. Сохранить файл с названием Пример1_2.html и открыть (запустить) его в любом браузере.

Использование переменных

В зависимости от того, для чего вы делаете скрипт, понадобится работать с информацией. ***Переменная*** **состоит из имени и выделенной области памяти, которая ему соответствует**.

Для *объявления* или, другими словами, *создания переменной* используется ключевое слово `var`:

```
var message;
```

После объявления, можно записать в переменную данные:

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

Эти данные будут сохранены в соответствующей области памяти и в дальнейшем доступны при обращении по имени:

```
var message;
message = 'Привет';
alert(message); // выведет содержимое переменной
```

Для краткости можно совместить объявление переменной и запись данных:

```
var message = 'Привет';
```
При изменении значения старое содержимое переменной удаляется.
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:
```
var num = 100500;
var message = 'Привет';
```
Значение можно копировать из одной переменной в другую.
```
var num = 100500;
var message = 'Привет';
message = num;
```
Значение из `num` перезаписывает текущее в `message`.
В JavaScript вы можете создать переменную и без `var`, 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 `var`. Это хороший тон в программировании и помогает избежать ошибок.
Пример документа с объявлением переменных:
```
<html>
<body>
  <div id="test"></div>

  <script>
    var test = 5;
    alert(test);
  </script>

</body>
</html>
```
Самое «забавное» — то, что, эта ошибка будет только в IE<9, и только если на странице присутствует элемент с совпадающим `id`.
Такие ошибки особенно весело исправлять и отлаживать.
Есть и еще ситуации, когда отсутствие `var` может привести к ошибкам. Надеюсь, вы убедились в необходимости всегда ставить `var`.

Задание: Создайте документ HTML, в котором средствами JavaScript:

1. Объявите две переменные: `admin` и `name`.
2. Запишите в `name` строку `"Василий"`.
3. Скопируйте значение из `name` в `admin`.
4. Выведите `admin` (должно вывести «Василий»).

Объявление констант

Константа — это переменная, которая никогда не меняется. Как правило, их называют большими буквами, через подчёркивание. Например:
```
var COLOR_BLUE = "#00F";
var COLOR_RED = "#0F0";
var COLOR_GREEN = "#F00";
```

```
var 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: `var`, `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.

Типы данных в JavaScript

Число `number`:

```
var n = 123;
n = 12.345;
```

Строка `string`:

```
var 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:

```
var 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`:

```
var 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`:

   ```
   var u;
   alert(u); // выведет "undefined"
   ```

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

   ```
   var 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`.

Пример объявления переменных и их использования:

```
<!DOCTYPE html>
<html>
<body>

<script>
```

```
var pi=3.14;
var name="John Doe";
var answer='Yes I am!';

document.write(pi + "<br>");
document.write(name + "<br>");
document.write(answer + "<br>");
</script>

</body>
</html>
```

**Здесь в скрипте
 - это тег HTML, который переводит каретку на новую строку (переход на новую строку)**

Задание: Создать приведенный пример документа в любом редакторе. Удобнее использовать редактор Visual Studio или любой другой, поддерживающий языки разметки. Сохранить файл с названием Пример1_3.html и открыть (запустить) его в любом браузере.

Пример

```
<!DOCTYPE html>
<html>
<body>

<p>Нажмите кнопку для объявления переменной и вывода результата.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo">Текст, помеченный меткой demo</p>

<script>
//объявляем функцию, которую будет вызывать кнопка по методу onclick
function myFunction()
{
//объявляем переменную строковую
var 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 устанавливает или получает всю разметку
// и содержание внутри данного элемента.

document.getElementById("demo").innerHTML=carname;
}
</script>
```

```
</body>
</html>
```

Задание: Создать приведенный пример документа в любом редакторе. Удобнее использовать редактор Visual Studio или любой другой, поддерживающий языки разметки. Сохранить файл с названием Пример1_4.html и открыть (запустить) его в любом браузере.

Пример на Undefined and Null:

```
<!DOCTYPE html>
<html>
<body>

<script>
var person;
var car="Volvo";

// document.write- метод, выводящий на страницу переданные ему аргументы

document.write(person + "<br>");
document.write(car + "<br>");
var car=null
document.write(car + "<br>");
</script>

</body>
</html>
```

Задание: Создать приведенный пример документа в любом редакторе. Удобнее использовать редактор Visual Studio или любой другой, поддерживающий языки разметки. Сохранить файл с названием Пример1_5.html и открыть (запустить) его в любом браузере.

Пример создает объект с названием "person" и добавляет 4 свойства объекту:

```
<!DOCTYPE html>
<html>
<body>

<script>
var person=new Object();
person.firstname="John";
person.lastname="Doe";
person.age=50;
person.eyecolor="blue";
document.write(person.firstname + " is " + person.age + " years old.");
</script>

</body>
</html>
```

Задание: Создать приведенный пример документа в любом редакторе. Удобнее использовать редактор Visual Studio или любой другой, поддерживающий языки

разметки. Сохранить файл с названием Пример1_6.html и открыть (запустить) его в любом браузере.

Основные 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 програмирования.

- *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. Далее мы увидим примеры.

2. Арифметические операторы

Базовые арифметические операторы знакомы нам с детства: это плюс +, минус -, умножить *, поделить /.

Например:

```
alert(2 + 2); // 4
```

Или чуть сложнее:

```
var i = 2;

i = (2 + i) * 3 / i;

alert(i); // 6
```

Более редкий арифметический оператор `%` интересен тем, что никакого отношения к процентам не имеет. Его результат `a % b` — это остаток от деления `a` на `b`.

Например:

```
alert(5 % 2); // 1, остаток от деления 5 на 2
alert(8 % 3); // 2, остаток от деления 8 на 3
alert(6 % 3); // 0, остаток от деления 6 на 3
```

3. 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
```

4. 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
```

5. Присваивание

Оператор присваивания выглядит как знак равенства =:

```
var i = 1 + 2;

alert(i); // 3
```

Он вычисляет выражение, которое находится справа, и присваивает результат переменной. Это выражение может быть достаточно сложным и включать в себя любые другие переменные:

```
var a = 1;
```

```
var b = 2;

a = b + a + 3;  // (*)

alert(a); // 6
```

В строке `(*)` сначала произойдет вычисление, использующее текущее значение `a` (т.е. `1`), после чего результат перезапишет старое значение `a`.

Возможно присваивание по цепочке:

```
var a, b, c;

a = b = c = 2 + 2;
```

Такое присваивание работает справа-налево, то есть сначала вычислятся самое правое выражение `2+2`, присвоится в `c`, затем выполнится `b = c` и, наконец, `a = b`.

Оператор "=" возвращает значение

Все операторы возвращают значение. Вызов `x = выражение` записывает выражение в `x`, а затем возвращает его. Благодаря этому присваивание можно использовать как часть более сложного выражения:

```
var a = 1;
var b = 2;

var c = 3 - (a = b + 1);

alert(a); // 3
alert(c); // 0
```

В примере выше результатом `(a = b + 1)` является значение, которое записывается в `a` (т.е. `3`). Оно используется для вычисления `c`.

Забавное применение присваивания, не так ли?

Знать, как это работает — стоит обязательно, а вот писать самому — только если вы уверены, что это сделает код более читаемым и понятным.

6. 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.

7. Инкремент/декремент: `++, --`

Одной из наиболее частых операций в JavaScript, как и во многих других языках программирования, является увеличение или уменьшение переменной на единицу.

Для этого существуют даже специальные операторы:

- **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.

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;
alert( 2 * i );
i++;
```

Важность: 5
Посмотрите, понятно ли вам, почему код ниже работает именно так?

```
var a = 1, b = 1, c, d;

c = ++a; alert(c); // 2
d = b++; alert(d); // 1

c = (2+ ++a); alert(c); // 5
d = (2+ b++); alert(d); // 4

alert(a); // 3
alert(b); // 3
```

8. 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) (`>>>`)

9. 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;

var x = 1 + (a *= 2);
```

10. 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:

```
<!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>
```

Задание: Создать приведенный пример документа в любом редакторе. Удобнее использовать редактор Visual Studio или любой другой, поддерживающий языки разметки. Сохранить файл с названием Пример1_7.html и открыть (запустить) его в любом браузере.

Взаимодействие с пользователем: alert, prompt, confirm

В этом разделе мы рассмотрим базовые UI операции: alert, prompt и confirm, которые позволяют работать с данными, полученными от пользователя.

Оператор alert

Синтаксис:

alert(сообщение)

alert выводит на экран окно с сообщением и приостанавливает выполнение скрипта, пока пользователь не нажмет «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 разберется с окном. В данном случае - пока не нажмет на «ОК».

Оператор 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 ввел посетитель - строку или специальное значение null, если ввод отменен.

Как и в случае с alert, окно prompt модальное.

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


alert('Вам ' + years + ' лет!')
```

Всегда указывайте default

Вообще, второй default может отсутствовать. Однако при этом IE вставит в диалог значение по умолчанию "undefined".

Запустите этот код в IE, чтобы понять о чем речь:

```
var test = prompt("Тест");
```

Поэтому рекомендуется *всегда* указывать второй аргумент:

```
var test = prompt("Тест", ''); // <-- так лучше
```

Оператор 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:

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


alert(isAdmin);
```

Место, где выводится модальное окно с вопросом, и внешний вид окна выбирает браузер. Разработчик не может на это влиять.

С одной стороны — это недостаток, т.к. нельзя вывести окно в своем 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.

Индивидуальные задания по вариантам

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 либо лично преподавателю, либо на электронную почту для проверки. Также по результатам лабораторной работы на следующем за ней занятии проводится выборочный опрос по командам языка.