ПРАКТИЧЕСКАЯ РАБОТА №1

Цель работы: освоить основные возможности языка JavaScript.

Условный оператор. Синтаксис: if (условие){ оператор1 [} else { оператор2] }

Если условие истинно, выполняется оператор1, иначе - оператор2.

Пример 1. Пример стандартного оператора if:

if (a == 1) window.alert(“1 найдено!”); a=0;

Нельзя путать оператор сравнения (==) с оператором присваивания (=).

Совместное использование логических и условных операторов. Очень часто необходимо проверить сразу несколько условий. Это можно сделать с помощью логических операций || (логическое ИЛИ) и && (логическое И).

Пример 2.

if (phone==””) window.alert(“Ошибка!”)

if (email==””) window.alert(“Ошибка!”)

С помощью логического оператора эти два оператора объединяются в один: if (phone==””|| email==””) window.alert(“Ошибка!”)

Оператор else – альтернативные действия оператора if, которые выполняются, если условие оператора if ложно.

Пример 3. Логическая переменная checkData получает значение true, если значение переменной data равно эталонному значению. В противном случае выводится сообщение об ошибке и значение checkData становится равным false.

.. .

etalon=10;

if (data == etalon) { checkData = true;}

else { alert("Произошла ошибка системы”); checkData = false;

} .. .

Оператор switch. При необходимости выбрать более чем из двух вариантов, удобно пользоваться оператором множественного выбора switch.

Оператор switch состоит из нескольких основных элементов:

Ключевое слово switch, за которым следует селектор выбора. Тело оператора switch заключается в фигурные скобки.

Один или несколько вариантов, определяемых словом case. Каждая такая строка содержит значение, сравниваемое со значением селектора. Если значения совпадают, операторы этого варианта выполняются. Если нет – пропускаются. Ключевое слово break используется для определения конца действия текущего варианта.

Необязательный оператор default выполняется, если ни один из вариантов не выполнен.

Пример 4.

switch (a)

{ case 1: window.alert(”1”); break;

case 2: window.alert(”2”); break; case 3: window.alert(”3”); break; default: window.alert(”Error!”);}

Организация циклических алгоритмов.

Цикл - автоматическое повторение выполнения одного или нескольких операторов, составляющих тело цикла.

Оператор цикла for. В операторе for цикл повторяется до тех пор, пока условие не станет ложным.

Синтаксис:

for ([**начальное выражение**];[**условие**];[**выражение приращения**]) {**тело цикла**}

Внутри тела цикла могут использоваться операторы break и continue, соответственно для выхода из цикла и выхода из текущей итерации цикла.

Пример 5.

for (i=1; i<10; i++) { document.write("Это строка No ",i,"
");}

Оператор цикла while. Цикл while выполняется до тех пор, пока указанное условие является истинным. Если условие сразу ложно, то цикл выполняться не будет.

Пример 6.

count=0;

value=3;

while (count<10) { count+=value; }

Оператор цикла do ... while. Единственное отличие от while – условие расположено в конце кода цикла.

Пример 7.

count=0;

value=3;

do { count+=value; } while (count<10);

В отличие от цикла с предварительным условием, данный цикл будет выполняться по крайней мере один раз, так как условие находится после тела цикла.

Функции

Функция - логически завершенный и определенным образом оформленный набор операторов, который выполняет некоторую задачу. Чтобы использовать функцию, необходимо сначала определить ее.

Определение функции состоит из ключевого слова function, за которым следуют:

- имя функции,
- в круглых скобках список аргументов функции, отделенных друг от друга запятыми,
- блок операторов JavaScript, заключенный в фигурные скобки.

Операторы могут включать в себя вызовы функций, определенных в текущем приложении. Различие между определением и вызовом функции традиционно для языков программирования. Определение функции просто называет функцию и задает выполняемые ею действия. Запрос функции исполняет указанные действия с фактическими параметрами. Лучше всего определять функции для страницы в разделе HEAD документа. В этом случае все функции будут определены прежде, чем показано содержание документа. Иначе, в то время как страница еще не полностью загружена, пользователь мог бы исполнить действие, которое вызывает еще не загруженную функцию, что привело бы к ошибке.

Пример 8. Простая функция:

```
function simplefun(str)
    { document.write("<HR><P>" + str); }
```

Эта функция получает строку str как аргумент, добавляет некоторые HTML-признаки к ней с помощью оператора конкатенации и выводит результат в текущем документе с помощью метода write.

Можно использовать любую функцию, определенную в текущей странице. Можно также использовать функции, определенные в других

поименованных окнах и фреймах. Для того, чтобы функция выполнилась ее необходимо вызвать. Предположим, функция simplefun была определена в разделе HEAD документа, тогда выполнить ее можно, например, следующим образом:

<SCRIPT language="JavaScript"> simplefun(”Вызвать функцию simplefun”); </SCRIPT>

Аргументы функции не ограничиваются строками и числами, можно передавать в качестве аргументов также и целые объекты. Функция может быть рекурсивной, то есть может вызывать сама себя. Рассмотрим функцию, которая вычисляет факториал.

Пример 9.

```
function factorial(n)
{ if (n<=1) { return 1;}
else { result = (n * factorial(n-1)); return result;
} }
for (x = 0; x < 5; x++) { document.write("<BR>", x, " factorial is ", factorial(x)); }
```

Результат:

0 factorial is 1

1 factorial is 1

2 factorial is 2

3 factorial is 6

4 factorial is 24 5 factorial is 120

Введение в Объектную Модель

С помощью объектной модели JavaScript взаимодействует с HTML страницей, загруженной в браузер, и может изменять и контролировать свойства окна просмотра и загруженной страницы.

В Объектной Модели браузера существуют два главных объекта:

- window – собственно окно просмотра,
- document – загруженная в это окно страница.

Чтобы увидеть это, следует загрузить в браузер какую-либо HTML-страницу и посмотреть на экран.

Все, что относится к браузеру: кнопки навигации, строка состояния, поле ввода адреса, меню и т.д. все это принадлежит объекту window.

Все, что относится к загруженной странице: графика, формы, ссылки, заголовок и т.д. – относится к объекту document.

Чтобы воспользоваться этими свойствами, получить к ним доступ, необходимо организовать ссылки на данные объекты. Например, чтобы записать что-либо в строку состояния нужно обратится к соответствующему объекту status:

window.status = “Запись в строке состояния”;

Объектная модель состоит из совокупности объектов, свойств, методов и событий. Объект – это одна из частей браузера, которая выполняет определенные задачи.

Например объект window отвечает за вид окна просмотра, за адрес, по которому загружен документ; объект document ответственен за загруженный документ; дочерний объект window – navigator в основном содержит информацию об агенте пользователя.

Свойства возвращают информацию о данном объекте. Например, через свойство объекта location можно узнать, какая страница загружена.

С помощью методов объекты выполняют возложенные на них функции. Например, методы объекта window позволяют открывать новые окна

Посредством событий объекты сообщают о каких-либо произошедших действиях. Например, событие для кнопки формы onClick сообщает, что кнопка была нажата.

Объект window

К объекту window относится открытое окно браузера. Объект window является главным объектом объектной модели браузера, – все остальные объекты являются дочерними объектами объекта window.

Свойства window представлены в таблице 1:

Свойство	Описание
defaultStatus	Текст по умолчанию в строке состояния
document	Ссылка на объект окна document
frames	Ссылка на коллекцию frames, содержащую все фреймы окна
history	Ссылка на объект окна history
length	Количество фреймов в родительском окне
location	Ссылка на объект окна location
name	Название окна
navigator	Ссылка на объект окна navigator
parent	Возвращает родительское окно для текущего
self	Возвращает ссылку на текущее окно
status	Текст в строке состояния
top	Ссылка на главное окно

Методы window

- **open** С помощью метода open() открываются новые окна просмотра.

Синтаксис: open(“URL_открываемого_окна”,”Имя_нового_окна”, [“Установки_нового_окна”])

Значения свойства Установки нового окна (WindowFeatures) приведены в таблице 2.

Значение	Описание
directories	Создает панель ссылок браузера
fullscreen	Разворачивает окно полностью на весь экран
height	Задает высоту окна
location	Создает поле ввода адреса
menubar	Создает стандартное меню
resizeable	Определяет, сможет ли пользователь изменять размер окна
scrollbars	Создает полосы прокрутки
status	Создает строку состояния
toolbar	Панели инструментов браузера
width	Задает ширину окна

Самый простой способ открыть окно, это применение метода open без параметров. Нужно просто указать адрес открываемой страницы:

function Open_Window() { window.open("New.htm"); }

В результате на экране появится новое окно, размер которого зависит от настроек используемого браузера.

Можно задать параметры явно: function Open_Window()

{ window.open("New.htm","My_Win","toolbar=yes,height=200,width= 600"); }

- **close** Закрывает текущее окно просмотра.

Пример 10.

<SCRIPT language="JavaScript"> function close_window() { close(); }

</ SCRIPT >

Не следует злоупотреблять методом close, поскольку будучи вызванным он может закрыть окно браузера без каких-либо предупреждений. Последние версии браузеров Netscape Navigator и MS Internet Explorer выдают предупреждение пользователю, с помощью которого он может отменить закрытие текущего окна.

- **alert** Выводит на экран окно сообщения. Синтаксис: alert (текст окна сообщения)
- **confirm** Выводит окно подтверждения.

Синтаксис:

- **confirm** (текст окна подтверждения)
- **prompt**

Синтаксис:

Например, prompt (“Введите ваше имя”, “Вводите сюда”);

Выводит окно ввода.

prompt (текст окна ввода, начальное значение поля ввода)

setTimeout Выполняет заданное выражение по истечении установленного времени (в миллисекундах).

Синтаксис:

setTimeout(expression, msec)

где expression – выражение, msec – задержка в миллисекундах.

Объект location

Объект location содержит информацию об адресе страницы (URL) и позволяет управлять им.

Свойство href объекта location содержит адрес страницы:

alert (“Данная страница находится по адресу: “ + location.href);

С помощью метода reload объекта location загруженная страница может быть обновлена. Эффект тот же, что и при нажатии кнопки Обновить.

Пример 11.

```
<SCRIPT language="JavaScript">
if (confirm(“Обновить страницу?”))
{ location.reload(); //обновляем страницу: alert(“Страница обновлена”);
}
else { alert(“Страница не обновлена!”); }
</SCRIPT >
```

Объекты Array

JavaScript не имеет типа данных, определяющих массив (array). Однако, можно использовать встроенный объект Array и его методы и работать с массивами в приложениях. Объект Array имеет методы для соединения, перевертывания и сортировки массивов. У него есть свойство для определения длины массива.

Создание массивов возможно двумя способами:

1. arrayObjectName = new Array([arrayLength]);

2. arrayObjectName = new Array([element0, element1, ..., elementn]);

arrayObjectName является или названием нового объекта, или свойством существующего объекта, arrayLength - начальная длина объекта. Можно получить доступ к этому значению используя свойство length; element0, element1,..., elementn - список значений для элементов массива, если использована вторая форма инициализации массива, его длина определяется количеством аргументов.

Объект Array имеет следующие методы:

- join связывает все элементы массива в строку;
- reverse переворачивает элементы массива: первый элемент становится последним и наоборот;
- sort сортирует элементы массива.

Пример 12.

Предположим, что мы определяем следующий массив: myArray = new Array("Один","Два","Три");

1. myArray.join() возвратит "Один,Два,Три";
2. myArray.reverse преобразует массив так, что myArray[0] есть"Три", myArray[1] есть "Два", и myArray[2] есть "Один".
3. myArray.sort сортирует элементы массива в лексикографическом порядке, так что myArray[0] есть "Два", myArray[1] есть "Один", и myArray[2] есть "Три".

Элементы массива можно определять с помощью оператора присваивания, например:

emp[1] = «Раз";

emp[2] = «Два";

emp[3] = "Три";

Допускается инициализация массива: myArray = new Array("Hello", myVar, 3.14159);

Пример 13. Создаем двумерный массив и выводим на экран. a = new Array(4);

for (i=0; i < 4; i++) { a[i] = new Array(4);

for (j=0; j < 4; j++) { a[i][j] = "["+i+","+j+"]"; } }

for (i=0; i < 4; i++) {

str = "Row «+i+":";

for (j=0; j < 4; j++) { str += a[i][j]; } document.write(str,"<P>");

}

В результате будет напечатано:

Row 0:[0,0][0,1][0,2][0,3] Row 1:[1,0][1,1][1,2][1,3] Row 2:[2,0][2,1 [2,2][2,3] Row 3:[3,0][3,1][3,2][3,3]

Можно обратиться к элементам массива, используя значение элемента или его индекс. Например, для массива myArray = new Array(” Один”,”Два”,”Три”); обращение к первому элементу возможно как myArray [0] или myArray ["Один"].

Объект Date

Работа с датой и временем, установленными на компьютере, в JavaScript осуществляется с помощью объекта Date (табл. 3):

let DateObject = new Date([значение]);

где параметр значение может принимать следующие значения:

- миллисекунды - количество миллисекунд прошедшее от 01/01/70 00:00:00; - год, месяц, день;
- год, месяц, день, часы, минуты, секунды;
- год, месяц, день, часы: минуты: секунды.

Метод	Описание
getDate()	Получить число месяца
getDay()	Получить номер дня недели
getHours()	Получить час
getMinutes()	Получить минуту
getMonth()	Получить номер месяца
getSeconds()	Получить секунду
getTime()	Получить количество миллисекунд между 1 января 1970 года и текущим временем
getTimezoneOffset()	Получить разницу в минутах между местным и гринвичским временем
getYear()	Получить год

setDate()	Установить день месяца
setHours()	Установить час
setMinutes()	Установить минуту
setMonth()	Установить месяц
setSeconds()	Установить секунду
setYear()	Установить год
toGMTString()	Преобразовать местное время во время по Гринвичу
toLocaleString()	Преобразовать время по Гринвичу в местное время
UTC()	Возвращает количество миллисекунд между 01/01/70 00:00:00 и заданным временем в формате объекта Date

Пример 14.

```
let D, day;
D = new Date();
day= D.getDate(); alert("Сегодня " + day + " число");
```

Объект Math

Встроенный объект Math имеет свойства и методы для математических констант и функций. Например, свойство PI объекта Math имеет значение 3.14159..., который можно использовать как Math.PI. Стандартные математические функции являются методами Math (табл. 4).

Метод	Описание
abs	абсолютное значение
sin, cos, tan	стандартные тригонометрические функции, аргумент в радианах

acos, asin, atan	обратные тригонометрические функции, возвращают значение в радианах
exp, log	экспонента и натуральный логарифм
ceil	возвращает наименьшее целое, большее или равное аргументу
floor	возвращает наибольшее целое, меньшее или равное аргументу
min, max	возвращает меньшее или большее (соответственно) из двух аргументов
pow	возведение в степень, первый аргумент основание, второй показатель степени
round	округляет аргумент до ближайшего целого
sqrt	квадратный корень

Например, если нужно использовать синус, следует писать Math.sin(1.56). Все тригонометрические методы Math используют аргументы в радианах.

Часто удобно использовать оператор with, когда есть раздел, в котором используется набор констант и методов, чтобы не повторять слово "Math":

```
with (Math) { a = PI * r*r; y = r*sin(alfa); x = r*cos(alfa); }
```

Объект document. Иерархия объектов страницы

Объект document содержит информацию о загруженном документе и отвечает за его отображение.

Коллекции объекта document

- **forms** Коллекция forms представляет собой массив, куда входят все формы текущей страницы. Каждый объект form в свою очередь содержит массив elements, куда входят все объекты формы, т.е. объекты расположенные между тегами <FORM> и </FORM>.

- **anchors** Коллекция anchors представляет собой массив, в котором находятся все метки (то есть списки внутри страницы) документа.

Синтаксис anchor в HTML-странице:

<A name = “anchorName”>anchorText</A>

- **images** Коллекция images содержит все рисунки, вставленные в документ, т.е. находящиеся в теге <IMG> , представляющие собой объект image. Каждый из них содержит коллекцию areas, содержащую выделенные области изображения. Свойство .src содержит url загруженного изображения

- **links** Коллекция links содержит все гиперссылки документа.

Доступ к коллекциям и элементам

JavaScript предоставляет несколько способов для доступа к ним.

1. Доступ через индекс массивов. Поскольку коллекции представляют собой массивы элементов, доступ к ним осуществляется как к элементам массивов. Так, если мы имеем массив links, первая встречающаяся гиперссылка в документе будет иметь индекс 0, вторая – 1, третья – 2 и т.д. Это справедливо для всех коллекций.

Пример 15. Страница, содержащая форму с полем ввода. <FORM name=”Forma1”>

<INPUT type=”text” value=”Вводить сюда” name=”Text1”>

<INPUT type=”button” value=”Показать” onClick=”ShowText()”> </ FORM>

Мы создали форму с полем ввода и кнопкой, вызывающей функцию ShowText(), которая будет показывать, какой текст введен в поле ввода. Для создания данной функции, нам нужно получить доступ к тексту, введенному в поле ввода. Делается это следующим образом. Известно, что все формы на странице входят в коллекцию forms, и каждая имеет свой порядковый номер. Наша форма единственная на странице, следовательно она будет первой, поэтому ее порядковый номер (индекс) будет равен 0. Поэтому, для получения доступа к нашей форме нужно составить следующую строку: document.forms[0] .

После того, как мы получили доступ к форме, нам нужно получить доступ к полю ввода. Как упоминалось выше, все элементы, включенные в форму, доступны с помощью коллекции elements. Так как поле ввода встречается первым, то его индекс будет ноль. Таким образом получаем строку

document.forms[0].elements[0] .

Данные, введенные в поле ввода, находятся в параметре value и полностью конструкция, содержащая данные строки ввода будет иметь следующий вид:

document.forms[0].elements[0].value.

2. Доступ через атрибут name

С помощью атрибута name каждой коллекции, а также ее элементам можно присвоить собственное имя для идентификации его сценарием.

Мы присвоили форме имя “Forma1” и строке ввода имя “Text1”. Кнопка из сценария не вызывается, поэтому присваивать ей имя не обязательно.

Для получения доступа к введенным данным из сценария, заменим указание коллекций присвоенными именами:

document.Forma1.Text1.value.

Свойства объекта **document** дают доступ к информации о загруженном документе (табл. 5).

Свойство	Описание
cookie	Выводит cookie, сохраненный браузером
lastModified	Дата последнего изменения страницы
location	Местонахождение страницы
parentWindow	Родительское окно (для фреймов)
referrer	Адрес страницы, из которой вызвана текущая страница
title	Заголовок документа
URL	Адрес страницы

Все эти свойства “только для чтения” т.е. их нельзя изменять из сценария.

С помощью свойства цвета объекта document (табл.6) мы можем получать и устанавливать значения цветового оформления страницы (фона, текста, гиперссылок и т.д.).

Свойство	Описание
bgColor	Выдает/устанавливает значение фона страницы в RGB- представлении.
fgColor	Возвращает/устанавливает цвет обычного текста.
linkColor	Цвет гиперссылок.
alinkColor	Цвет активных гиперссылок.
vlinkColor	Цвет посещенных гиперссылок.

Пример 16:

```
<SCRIPT>
function Change_bgcolor()
{ let Ccolor = prompt( "Введите новый цвет для фона страницы:»,
```

""); document.bgColor = Ccolor;

}

</ SCRIPT >

При запуске данного сценария можно ввести значение цвета RGB-кодом, или его английским названием, например, green.

Методы объекта document (табл. 7) используются для работы с самим содержанием документа, для его изменения и редактирования.

Свойство	Описание
open()	Открывает документ для записи
write()	Записывает данные в HTML-документ
close()	Закрывает документ
clear()	Удаляет содержимое страницы

- **open** Открывает HTML-документ для записи. Чтобы в документ что-либо записать, его нужно сначала открыть методом open(). Нельзя путать данный метод, с методом open() объекта window. Метод window.open() служит для открытия нового окна просмотра и может быть записан просто open(), а данный метод служит для открытия документа для записи и, чтобы было ясно, что это метод document, нужно обязательно ссылаться на него: document.open();
- **write** Позволяет записывать данные в документ.

Пример 17.

<HTML>

<HEAD><TITLE>Использование метода write()</TITLE > </HEAD >

<BODY>

<SCRIPT>

document.write(“Эта надпись будет видна только в браузерах,
 поддерживающих JavaScript.”);

</SCRIPT>

</BODY >

</HTML>

Как видно из примера, при помощи write() можно вставлять также тэги HTML. Вместе с методом write() существует также метод writeln(), который отличается тем, что в конце записываемой строки он автоматически добавляет в конце строки переход на новую строку. Эта разница практически незаметна, кроме нескольких случаев.

Пример 18.

Использование методов объекта document в элементе <PRE>, где сохраняется разбиение на строки:

```
<SCRIPT>
document.writeln("<PRE>Это первая строка");
document.writeln("Это вторая строка");
document.writeln("а это...");
document.writeln("четвертая строка</PRE>");
</SCRIPT>
```

- **close** Метод close() закрывает открытую методом open() страницу. Как и метод open(), не следует данный метод смешивать с методом close() объекта window. Нужно всегда перед ним ставить ссылку на объект document: document.close(); так как иначе закроется все окно.
- **clear** Метод clear, применяется для очистки содержимого страницы, т.е. для удаления всего кода HTML. Для его использования необходимо сначала открыть страницу методом document.open(), а затем закрыть методом document.close().

Пример 19.

```
<SCRIPT>
function clear_doc()
{ if (confirm("Очистить документ?"))
{ document.open(); document.clear(); document.close();
} }
```

</SCRIPT>

//открываем документ для редактирования //очищаем документ

//закрываем его

Включение JavaScript в HTML- документы

Существуют следующие способы подключения JavaScript-программ к HTML- документу:

1. Использование элемента <SCRIPT>.
2. Объявление JavaScript-файла, содержащего текст программы.
3. Объявление JavaScript-программы в качестве обработчика событий.

Использование элемента SCRIPT. Элемент <SCRIPT> может включать любое число JavaScript-операторов:

<SCRIPT> JavaScript - операторы ... </SCRIPT>

Документ может иметь любое количество элементов SCRIPT.

Определение файла для JavaScript. Атрибут src элемента <SCRIPT> позволяет определять файл как источник операторов JavaScript.

Пример 20.

<HEAD>

<TITLE>My Page</TITLE>

<SCRIPT src="myprog1.js">

alert(”Файл, содержащий скрипт, не подключился!”); </SCRIPT>

</HEAD>

<BODY>

…

</BODY>

Внешние файлы JavaScript должны содержать только JavaScript-определения функций и операторы, в них не может быть HTML-тегов. Внешние файлы JavaScript должны иметь расширение .js.

Пример 21.

<HEAD>

<SCRIPT language="JavaScript">

<!-- Скрываем от старых браузеров function myfunc(number) { return number + number;} // Конец сокрытия -->

</SCRIPT>

</HEAD>

<BODY>

<SCRIPT language="JavaScript">

document.write("Удвоенный аргумент = ", myfunc(7), «.");

</SCRIPT>

<P> OK ! </BODY>

Функция myfunc имеет один аргумент, названный number. Состоит она из одного оператора return number+number, который должен возвратить удвоенное значение аргумента функции. Вызывается функция с аргументом 7.

Результат, выводимый на экран, выглядит следующим образом:

Удвоенный аргумент = 14. OK!

Использование write-метода. С помощью write-метода можно выводить информацию в окно навигатора так же, как и с помощью HTML-операторов, но write-метод обладает большими возможностями, например можно работать с переменными аргументами. По этим причинам, write - один из наиболее часто используемых JavaScript-методов. write-метод работает с любым числом строковых аргументов, которые могут быть строковыми литералами или переменными. Можно также использовать строковую конкатенацию - оператор (+), чтобы создать одну строку из нескольких.

Пример 22.

Динамическое формирование HTML-страницы. <HEAD>

<SCRIPT>

<!-- Скрываем от старых браузеров

// Эта функция выводит горизонтальную линию заданной ширины

```
function bar(widthPct)
{ document.write("<HR align=”left” width=”widthPct” + ”%”>);}
// Эта функция показывает заголовок указанного уровня и некоторый текст
function output(headLevel, headText, text)
{ document.write(”<H”, headLevel, ”>”, headText, ”</H”, headLevel,”><P>”, text);
}
// Конец сокрытия-->
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
<!-- Начало сокрытия
bar(25);
output(2,"Это заголовок 2-го уровня", "Здесь можно добавить любую строку»);
// Конец сокрытия -->
</SCRIPT>
<P> Это стандартный HTML, в отличие от предыдущей строки, которая сгенерирована. </BODY>
```

Раздел HEAD этого документа определяет две функции:

- bar - функция выводит горизонтальную линию заданной ширины
- output - функция показывает заголовок указанного уровня и некоторый текст Раздел BODY вызывает обе функции с фактическими параметрами.

Задания

Посмотреть порядок работы примеров, представленных в лабораторной работе. Изменить содержание примеров по своему усмотрению с применением рассмотренных функциональных возможностей JavaScript.

1. Пользуясь теми основами языка JavaScript, которые были изложены в данной лабораторной работе, и знаниями HTML, написать код, содержащий функции JavaScript и вырисовывающий HTML страницу с различными тегами.
2. Написать функции, в которых использовались бы математические функции. Результат вычислений вывести в строку состояния.
3. Написать функции с использованием массивов Array.
4. Создать кнопку, выводящую сообщение с url изображения, загруженного на странице.
5. Вывести сообщение о дате последнего изменения загруженного документа.
6. Выдать окно запроса имени пользователя и вывести его в строку состояния.
7. Вывести в документ текущую дату, с использованием объекта Date.
8. Вывести текущее время, с использованием объекта Date.
9. Вывести разницу в минутах между местным и гринвичским временем.
10. Вывести заголовок загруженного документа.
11. Задать цвет фона для документа.
12. Задать цвет текста для документа.
13. Задать цвет гиперссылок в документе.

14. Осуществить переход по гиперссылке при нажатии соответствующей кнопки.

15. Внутри фрейма вывести имя родительского окна.

16. *Произвести сортировку произвольного числового массива, минуя метод .sort, методом “пузырька” и вывести результат в строку состояния.

Форма отчета

Отчет предоставляется в печатном виде с титульным листом установленного образца с приложением кода и скриншотов выполнения работы.