**ЛАБОРАТОРНАЯ РАБОТА №6.**

**Основы динамических web-страниц HTML, DHTML, JavaScript.**

Цель: Получить навыки разработки Web-страниц с использованием DHTML.

**HTML.**

HTML (от [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *HyperText Markup Language* — «язык [гипертекстовой](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%BF%D0%B5%D1%80%D1%82%D0%B5%D0%BA%D1%81%D1%82) разметки») — стандартный [язык разметки](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D1%80%D0%B0%D0%B7%D0%BC%D0%B5%D1%82%D0%BA%D0%B8) документов во [Всемирной паутине](https://ru.wikipedia.org/wiki/%D0%92%D1%81%D0%B5%D0%BC%D0%B8%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D1%83%D1%82%D0%B8%D0%BD%D0%B0). Большинство [веб-страниц](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0) содержат описание разметки на языке HTML (или [XHTML](https://ru.wikipedia.org/wiki/XHTML)).

* **HeperText -**  это текст, содержащий гиперссылки, связывающие слова или картинки документа с другим ресурсом (с каким-нибудь еще документом или с иным разделом этого же документа). **(Hyper** противоположно linear/построчно. Ранее компьютерные программы работали построчно: программа выполняла одну строку, затем переходила к выполнению следующей, и т. д. Но HTML работает по-иному — вы можете перейти куда и когда захотите. Например, не нужно посещать MSN.com до того, как посетить HTML.net).
* **Mark-up** — это разметка, то, что вы делаете с текстом. Вы размечаете текст так же, как вы делаете это в текстовых редакторах: выставляете заголовки, списки, выделяете текст жирным шрифтом и т. д.
* **Language** это язык — HTML.

HTML был изобретён в 1990 году учёным, Тимом Бёрнсом-Ли (Tim Berners-Lee), и предназначался для облегчения обмена документами между учёными различных университетов. Проект имел больший успех, чем Tim Berners-Lee мог ожидать. Этим изобретением HTML он заложил основы современной сети Internet.

HTML это язык, который позволяет представлять информацию (например, научные исследования) в Internet. То, что вы видите при просмотре страницы в Internet, это интерпретация вашим браузером HTML-текста. Чтобы увидеть HTML-коды страницы в Internet, щёлкните «View» в линейке меню вашего браузера и выберите «Source».

**Особенности языка HTML:**

1. Делает страницу **видимой** для различных Web-броузеров.
2. Задаёт **стиль** шрифтов, заголовков, фрейм-компонентов, форматирует текст.
3. Включает в Web-страницу **таблицы**, **картинки**, элементы **мультимедиа**.
4. Создаёт из текста и из графического или анимированного изображения **гиперссылки**, указывающие на другие Web -документы.
5. Имеет возможность создания **карт-изображений** с активными областями, т.е. создания множественной гиперссылки.
6. Имеет возможность включения **интерактивных компонентов**: бланков, анкетных и опросных форм, требующих участия пользователя.
7. Способен нести в себе элементы других языков: **XSL**, **Java**, **JavaScript** и т.д.

Структура HTML состоит из 3-х пар тегов:

*<html>*

*<head>*

Заголовок документа

*</head>*

*<body>*

Тело документа

*</body>*

*</html>*

Теги *<html></html>* являются контейнером для всех остальных, т.е. в них помещаются все остальные теги. Таким образом, ваш документ должен начинаться с тега *<html>,* а заканчиваться тегом *</html>*. Сам документ разделён на две части – заголовок документа(теги *<head></head>)* и тело документа(теги *<body></body>*).

**Заголовок документа – элементы тега HEAD.**

Заголовок документа содержит служебную информацию и не влияет на внешний вид документа. Его задачей является предоставление браузеру пользователя или серверу информации о том, как отобразить ваш документ.

*Тег TITLE.*

*<title></title>* является единственным обязательным элементом заголовка документа. Эти теги необходимы, чтобы дать документу название, оно отображается в заголовке окна браузера.

*<html>*

*<head>*

*<title>*Заголовок документа*</title>*

*</head>*

*<body>*

Тело документа

*</body>*

*</html>*

В браузере это будет выглядеть следующим образом:

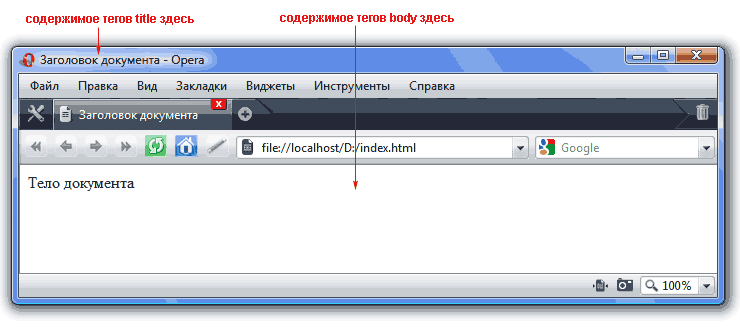


Рис1. Заголовок документа.

*Тег BASE.*

Одиночный тег *<base>* служит для указания полного URL-адреса документа. У тега есть один атрибут href, значение которого является адрес страницы.

*<html>*

*<head>*

*<title>*Заголовок документа*</title>*

*<base href = “http://www.my\_site.com/”>*

*</head>*

*<body>*

Тело документа

*</body>*

*</html>*

*Тег LINK.*

Одиночный тег *<link>* необходим для подключения внешних файлов. У тега несколько атрибутов:

* href – указывает URL-адрес подключаемого файла.
* rel – указывает на тип отношения данного документа к внешнему(например: rel=”stylesheet” указывает, что внешний файл определяет стиль документа).
* type – указывает тип и параметры присоединенной таблицы стилей.

*Тег META.*

Информация в этом теге не имеет никакого отношения к HTML, однако её использование очень удобно для решения ряда задач:

* Указания кодировки страницы.
* Указания ключевых слов страницы.
* Указания краткого описания страницы.

Параметры тега *META*:

* http-equiv – определяет свойство тега(тип контента, ключевые слова, описание и т.д.)
* name – используется для дополнительного описания тега, если отсутствует, то считается эквивалентным параметру http-equiv
* content – значение параметра http-equiv.

*<html>*

*<head>*

*<title>*Заголовок документа*</title>*

*<base href = “http://www.my\_site.com/”>*

*<link rel = stylesheet href=”styles/style.css” type=text/css>*

*<meta http-equiv=”Content-Type” content=”text/html” charset=”utf-8” />*

*</head>*

*<body>*

Тело документа

*</body>*

*</html>*

Тело документа *<body>* предназначено для размещения тегов и содержательной части.

Внутри контейнера *<body>* могут размещаться практически любые элементы веб-страницы.

Чтобы браузер понимал, что имеет дело не с простым текстом, а с особым элементом, который задает его форматирование, и применяются теги. Теги являются основой HTML и берутся в угловые скобки. Общий синтаксис написания тегов следующий.

*<тег параметр1=”значение” параметр2=”значение”>…</тег>*

Теги бывают одиночными и парными (контейнеры). Одиночный тег используется самостоятельно, а парный может включать внутри себя другие теги или текст. У тегов допустимы различные параметры, которые разделяются между собой пробелом. Впрочем, есть теги, без всяких дополнительных атрибутов. Параметры условно можно подразделить на обязательные, они непременно должны присутствовать, и необязательные, их добавление зависит от цели применения.

Любые теги, а также их параметры нечувствительны к регистру, поэтому форму записи вы вольны выбирать сами, как писать — *<BR>* или *<br>.* В любом случае рекомендуется придерживаться выбранной формы записи на протяжении всех страниц сайта.

Когда для тега не добавлен какой-либо допустимый параметр, это означает, что браузер в этом случае будет подставлять значение, заданное по умолчанию. Если вы ожидали получить иной результат на веб-странице, проверьте, возможно, следует явно указать значения некоторых параметров.

Согласно спецификации HTML все параметры тегов следует брать в двойные или одинарные кавычки. Отсутствие кавычек не приведет к ошибкам, браузеры достаточно корректно обрабатывают код и без кавычек, за исключением текста, содержащего пробелы.

Если какой-либо тег или его параметр был написан неверно, то браузер проигнорирует подобный тег и будет воспроизводить текст так, словно тега и не было.

Существует определенная иерархия вложенности тегов. Например, метатеги должны находиться внутри контейнера *<HEAD>* и нигде иначе. Чтобы не возникло ошибки, следите за тем, чтобы теги располагались в коде правильно.

Если теги между собой равноценны в иерархии связи, то их последовательность не имеет значения. Так, можно поменять местами теги *<TITLE>* и *<META>,* на конечный результат это никак не скажется.

*Закрывайте все теги!*

Существует три состояния закрывающего тега: обязателен, не требуется или не обязателен. Обязательный закрывающий тег должен присутствовать всегда, иначе это приведет к ошибке при отображении документа. Для некоторых тегов вроде *<BR>* закрывающего тега нет в принципе. Необязательный закрывающий тег говорит о том, что разработчик может его как добавить, так и опустить, к ошибке это не приведет. Однако рекомендуем закрывать все теги, включая необязательные, это дисциплинирует, создает более стройный и строгий код, который легко модифицировать.

Порядок параметров в любом теге не имеет значения и на результат отображения элемента не влияет.

### DynamicHTML.

**Dynamic HTML** или **DHTML** — это способ (подход) создания интерактивного [веб-сайта](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B0%D0%B9%D1%82), использующий сочетание статичного языка разметки [HTML](https://ru.wikipedia.org/wiki/HTML), встраиваемого (и выполняемого на стороне клиента) [скриптового языка](https://ru.wikipedia.org/wiki/%D0%A1%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) [JavaScript](https://ru.wikipedia.org/wiki/JavaScript),[CSS](https://ru.wikipedia.org/wiki/CSS) (каскадных таблиц стилей) и [DOM](https://ru.wikipedia.org/wiki/Document_Object_Model) (объектной модели документа).

Он может быть использован для создания приложения в [веб-браузере](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%B1%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80): например для более простой навигации или для придания интерактивности форм. DHTML может быть использован для динамического перетаскивания элементов по экрану. Также он может служить как инструмент для создания основанных на браузере видеоигр.

Что же дает DHTML разработчику Веб-узлов? По нашему мнению, основных преимуществ три:

1. *Повышение конкурентоспособности узла*. Узел, содержащий динамически изменяемые цвета, анимированную графику, систему выпадающих меню и другие стандартные (для современных ОС, но не для Веба) элементы пользовательского интерфейса, явно привлекательнее для пользователя, чем обычные статические страницы. Разумеется, при этом нужно иметь чувство меры и соблюдать общие принципы дизайна.
2. *Легкость сопровождения*. DHTML позволяет нам вынести типовые элементы формирования страниц нашего Веб-узла в отдельные таблицы стилей и сценарии, а затем включать их одинаковым образом во все страницы. Это, безусловно, упрощает сопровождение и минимизирует количество возникающих опечаток и других ошибок.
3. *Уменьшение трафика*. Одна из основных особенностей DHTML состоит в том, что он переносит основную тяжесть по формированию Веб-страниц с сервера на клиент. Платой за это является загрузка с сервера сценариев и программных компонентов, которые при правильном проектировании незначительны по объему. В результате динамические страницы загружаются и отображаются намного быстрее статических, что особенно важно для отечественных каналов связи, не страдающих чрезмерно высоким качеством.

**JavaScript.**

Что такое JavaScript?

JavaScript изначально создавался для того, чтобы сделать web-странички «живыми». Программы на этом языке называются скриптами. В браузере они подключаются напрямую к HTML и, как только загружается страничка — тут же выполняются.

**Программы на JavaScript — обычный текст**. Они не требуют какой-то специальной подготовки.

В этом плане JavaScript сильно отличается от другого языка, который называется [Java](http://ru.wikipedia.org/wiki/Java).

Когда создавался язык JavaScript, у него изначально было другое название: «LiveScript». Но тогда был очень популярен язык Java, и маркетологи решили, что схожее название сделает новый язык более популярным.

Планировалось, что JavaScript будет эдаким «младшим братом» Java. Однако, история распорядилась по-своему, JavaScript сильно вырос, и сейчас это совершенно независимый язык, со своей спецификацией, которая называется [ECMAScript](http://en.wikipedia.org/wiki/ECMAScript), и к Java не имеет никакого отношения.

У него много особенностей, которые усложняют освоение, но по ходу учебника мы с ними разберемся.

JavaScript может выполняться не только в браузере, а где угодно, нужна лишь специальная программа —[интерпретатор](http://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%B0%D1%82%D0%BE%D1%80). Процесс выполнения скрипта называют «интерпретацией».

**Компиляция и интерпретация, для программистов**

Для выполнения программ, не важно на каком языке, существуют два способа: «компиляция» и «интерпретация».

* *Компиляция* — это когда исходный код программы, при помощи специального инструмента, другой программы, которая называется «компилятор», преобразуется в другой язык, как правило — в машинный код. Этот машинный код затем распространяется и запускается. При этом исходный код программы остаётся у разработчика.
* *Интерпретация* — это когда исходный код программы получает другой инструмент, который называют «интерпретатор», и выполняет его «как есть». При этом распространяется именно сам исходный код (скрипт). Этот подход применяется в браузерах для JavaScript.

Современные интерпретаторы перед выполнением преобразуют JavaScript в машинный код или близко к нему, оптимизируют, а уже затем выполняют. И даже во время выполнения стараются оптимизировать. Поэтому JavaScript работает очень быстро.

Во все основные браузеры встроен интерпретатор JavaScript, именно поэтому они могут выполнять скрипты на странице. Но, разумеется, JavaScript можно использовать не только в браузере. Это полноценный язык, программы на котором можно запускать и на сервере, и даже в стиральной машинке, если в ней установлен соответствующий интерпретатор.

***Что умеет JavaScript?***

Современный JavaScript — это «безопасный» язык программирования общего назначения. Он не предоставляет низкоуровневых средств работы с памятью, процессором, так как изначально был ориентирован на браузеры, в которых это не требуется.

Что же касается остальных возможностей — они зависят от окружения, в котором запущен JavaScript. В браузере JavaScript умеет делать все, что относится к манипуляции со страницей, взаимодействию с посетителем и, в какой-то мере, с сервером:

* Создавать новые HTML-теги, удалять существующие, менять стили элементов, прятать, показывать элементы и т.п.
* Реагировать на действия посетителя, обрабатывать клики мыши, перемещение курсора, нажатие на клавиатуру и т.п.
* Посылать запросы на сервер и загружать данные без перезагрузки страницы(эта технология называется «AJAX»).
* Получать и устанавливать cookie, запрашивать данные, выводить сообщения…
* …и многое, многое другое!

***Что не умеет JavaScript?***

JavaScript — быстрый и мощный язык, но браузер накладывает на его исполнение некоторые ограничения..

Это сделано для безопасности пользователей, чтобы злоумышленник не мог с помощью JavaScript получить личные данные или как-то навредить компьютеру пользователя.

Этих ограничений нет там, где JavaScript используется вне браузера, например на сервере. Кроме того, современные браузеры предоставляют свои механизмы по установке плагинов и расширений, которые обладают расширенными возможностями, но требуют специальных действий по установке от пользователя

**Большинство возможностей JavaScript в браузере ограничено текущим окном и страницей.**

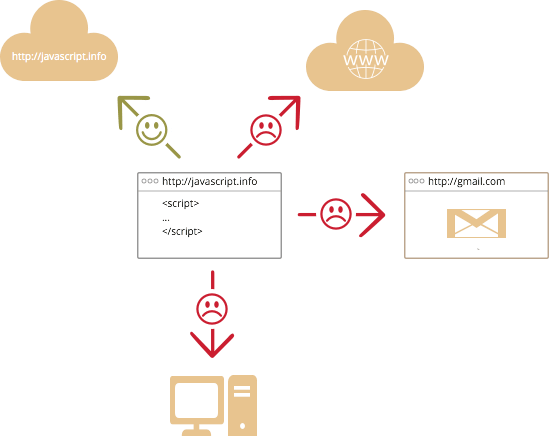


Рис.2. Возможности JavaScript.

* JavaScript не может читать/записывать произвольные файлы на жесткий диск, копировать их или вызывать программы. Он не имеет прямого доступа к операционной системе.

Современные браузеры могут работать с файлами, но эта возможность ограничена специально выделенной директорией — *«песочницей»*. Возможности по доступу к устройствам также прорабатываются в современных стандартах и частично доступны в некоторых браузерах.

* JavaScript, работающий в одной вкладке, не может общаться с другими вкладками и окнами, за исключением случая, когда он сам открыл это окно или несколько вкладок из одного источника (одинаковый домен, порт, протокол).
* Из JavaScript можно легко посылать запросы на сервер, с которого пришла страница. Запрос на другой домен тоже возможен, но менее удобен, т.к. и здесь есть ограничения безопасности.

## ***[В чем уникальность JavaScript?](https://learn.javascript.ru/intro" \l "в-чем-уникальность-javascript)***

Есть как минимум *три* замечательных особенности JavaScript:

* Полная интеграция с HTML/CSS.
* Простые вещи делаются просто.
* Поддерживается всеми распространенными браузерами и включен по умолчанию.

**Этих трёх вещей одновременно нет больше ни в одной браузерной технологии.**

Поэтому JavaScript и является самым распространенным средством создания браузерных интерфейсов.

Синтаксис JavaScript сделан по образцу C и C++. Отметим основные правила:

* **Чувствительность к регистру**. Все ключевые слова пишутся в нижнем регистре. Все переменные и названия функций пишутся точно так же, как и были определены (например, переменные Str и str являются разными переменными).
* **Пробелы, табуляция и перевод строки**. Эти символы игнорируются в JavaScript, так что можно использовать их для форматирования кода с тем, чтобы его было удобно читать.
* **Символ точка с запятой (;)**. Все операторы должны быть разделены этим символом. Если оператор завершается переводом строки, то точку с запятой можно опустить. При этом нужно следить за тем, чтобы при разрыве строки одного оператора, новая строка не начиналась бы с самостоятельного оператора.
* **Комментарии**. JavaScript игнорирует любой текст расположенный между символами /\* и \*/. Также игнорируется текст начинающийся символами // и заканчивающийся концом строки.
* **Индентификаторы**. Индентификаторами являются имена переменных, функций, а также меток. Индентификаторы образуются из любого количества букв ASCII, подчеркивания (\_) и символа доллара ($). Первым символом не может быть цифра, а в версии JavaScript 1.0 не допускается использования и символа $.
* **Ключевые слова**. Ключевые слова не могут использоваться в качестве индетификаторов. Ключевыми словами являются: break, case, continue, default, delete, do, else, export, false, for, function, if, import, in, new, null, return, switch, this, true, typeof, with.

## **Вставка в документ HTML**

Вы наверняка видели в HTML-кодах такие тэги:

<script>  
 …   
 </script>

Это и есть скрипт.

Сам тэг **<script>** относится к языку HTML, и у него могут быть следующие атрибуты:

* language
* type
* src

<script language=”javascript”>  
 код скрипта  
 </script>

Этот атрибут необязателен. Его стоит использовать либо для уточнения версии языка (javascript1.1, javascript1.2 и т.п.), либо если используешь другой язык (например, **<script language=Vbscript»>**). То есть вреда этот атрибут в любом случае не принесёт, но в стандартной ситуации он вроде как лишний.

<script type=”text/javascript”>  
 код скрипта  
 </script>

Атрибут type, который указывает на тип текста: **text/javascript**. Он появился в версии HTML 4.0.

Прежде чем перейти к атрибуту **src**, выясним, в какие разделы кода HTML можно вставлять скрипты.

В любые. Но с умом!

Часто в скрипте указывается вывод конкретного текста, что называется, здесь и сейчас. Такой скрипт вставляется прямо в *<body>,* «на место происшествия».

Бывают скрипты с объявлениями переменных, которые могут быть использованы в других скриптах на странице, с функциями, которые можно вызвать из любого места кода HTML. Такие скрипты разумнее всего помещать между тэгами *<head>* и *</head>.*

Но можно сделать и так, чтобы использовать скрипт сразу на нескольких web-страницах. Для этого его код нужно записать в отдельный файл с расширением **.js** (например, **myscript\_1.js**). Тэги *<script>* и *</script>* писать в нём уже не надо.

И вот тогда-то, для вызова скрипта с web-страницы, нам и понадобится атрибут **src**. Работает он так же, как и аналогичный атрибут тэга **<img>**:

<script type=”text/javascript” src=”scripts/myscript\_1.js”></script>

Вот таким образом и помещается на разные страницы одна и та же шапка или меню, записанные в файле скрипта.

Можно также вставлять маленькие скрипты в некоторые атрибуты тэгов HTML.

Пример 5.1.

Высчитаем сумму двух чисел, результат выведем на экран при помощи функции alert()(в примере сконцентрированы многие ключевые понятия языка javascript и его синтаксиса):

<html>

<head>

<title>Пример</title>  
</head>  
<body>

<script type=”text/javascript”>  
var х = 5; // *Объявляем переменную x, которая равна 5.*  
var y = х + 3; // *Объявляем переменную у.*  
**alert**(y); // *Вызов метода alert();*  
</script>

</body>  
</html>

В результате мы увидим следующее окно:

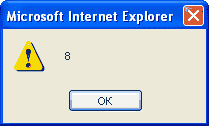


Рис.7. Результат программы.

**var** — ключевое слово для объявления переменной (по-английски variable).

***Ограничения:****В имени переменной можно использовать только латинские буквы (любого регистра), цифры и символ подчёркивания. При этом переменная не должна начинаться с цифр. И никаких пробелов.*

В этом скрипте переменной сразу при объявлении присваивается значение. Это не обязательно. Значение можно присвоить и позже. В конце строки стоит точка с запятой. Это в данном случае тоже не обязательно. Но в больших и сложных скриптах это иногда оказывается важным.

Метод *alert()* выводит на экран окно диалога с сообщением, указанным в скобках. Это синтаксис всех методов JavaScript: имя метода и скобки с его содержимым.

***Функции в JavaScript.***

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, еще где-нибудь.

Чтобы не повторять один и тот же код во многих местах, придуманы функции. Функции являются основными «строительными блоками» программы.

Пример встроенной функции вы уже видели — это alert(message). Но можно создавать и свои.

function showMessage(){

alert( ‘Hello world!’);

}

Вначале идет ключевое слово function, после него имя функции, затем список параметров в скобках (в примере выше он пустой) и тело функции — код, который выполняется при её вызове.

Объявленная функция доступна по имени, например:

showMessage();

Функция может содержать локальные переменные, объявленные через var. Такие переменные видны только внутри функции:

function showMessage(){

var message = ‘Hello world!’;

alert(message);

}

showMessage(); *// ‘Hello world!’;*

alert(message); *// ОШИБКА, т.к. переменная видна только внутри функции.*

**Блоки if/else, switch, for, while, do..while не влияют на область видимости переменных.**

При объявлении переменной в таких блоках, она всё равно будет видна во всей функции.

function count(){

*//переменные i, j не будут уничтожены по окончанию цикла*

for (var i = 0; i <3; i++){

var j=i\*2;

}

alert(i); *// i=3, последнее значение i, при нём цикл перестаёт работать*

alert(j); *// j=4, последнее значение j, которое вычислил цикл*

**Неважно, где именно в функции и сколько раз объявляется переменная. Любое объявление срабатывает один раз и распространяется на всю функцию.**

Функция также может обратиться и ко внешней переменной, например:

var username = ‘world!’;

function showMessage(){

var message = ‘Hello, ‘ + username;

alert(message);

}

showMessage(); *// Hello, world!*

alert( username); // *world!*

**Переменные, объявленные на уровне всего скрипта, называют «глобальными переменными».**

В примере выше переменная userName — глобальная.

Делайте глобальными только те переменные, которые действительно имеют общее значение для вашего проекта, а нужные для решения конкретной задачи — пусть будут локальными в соответствующей функции.

При вызове функции ей можно передать данные, которые та использует по своему усмотрению.

Например, этот код выводит два сообщения:

function showMessage(from, text){*// параметры from, text*

from = “\*\*” + from + “\*\*”; *// здесь может быть сложный код оформления*

alert(from + ‘: ‘ + text);

}

showMessage(‘Маша’, ‘Привет!’);

showMessage(‘Маша’, ‘Как дела?’);

Функция может возвратить результат, который будет передан в вызвавший её код.

**Для возврата значения используется директива return.**

Она может находиться в любом месте функции. Как только до нее доходит управление — функция завершается и значение передается обратно.

Вызовов return может быть и несколько, например:

function checkAge(age){

if(age>18){

return true;

} else {

return confirm(‘Позволяет ли возраст?’);

}

}

var age = promt(‘ Ваш возраст? ’); *// Пользователь должен либо что-то ввести и нажать OK, либо отменить ввод кликом на CANCEL или нажатием Escна клавиатуре.*

If (checkAge(age)){

alert(‘Доступ разрешён.’);

} else {

alert(‘В доступе отказано. ’);

}

# ***Браузерные события.***

Событие — это сигнал от браузера о том, что что-то произошло. Существует много видов событий. Посмотрим список самых часто используемых:

**События мыши**

* click — происходит, когда кликнули на элемент левой кнопкой мыши
* contextmenu — происходит, когда кликнули на элемент правой кнопкой мыши
* mouseover — возникает, когда на элемент наводится мышь
* mousedown и mouseup — когда кнопку мыши нажали или отжали
* mousemove — при движении мыши

**События на элементах управления**

* submit — посетитель отправил форму <form>
* focus — посетитель фокусируется на элементе, например нажимает на <input>

**Клавиатурные события**

* keydown — когда посетитель нажимает клавишу
* keyup — когда посетитель отпускает клавишу

**События документа**

* DOMContentLoaded — когда HTML загружен и обработан, DOM документа полностью построен и доступен.

**События CSS**

* transitionend — когда CSS-анимация завершена.

Также есть и много других событий.

Событию можно назначить обработчик, то есть функцию, которая сработает, как только событие произошло.

Именно благодаря обработчикам JavaScript -код может реагировать на действия посетителя.

Есть несколько способов назначить событию обработчик.

*Использование атрибута HTML.*

Обработчик может быть назначен прямо в разметке, в атрибуте, который называется on<событие>.

Например, чтобы прикрепить click-событие к input кнопке, можно присвоить обработчик onclick, вот так:

*<input value=”Нажми меня” onclick=”alert(‘Клик!’)” type=”button”>*

При клике мышкой на кнопке выполнится код, указанный в атрибуте onclick.

Пример 5.2.

Пример реализует запуск функции countRabbits() по клику, результатом которой является подсчёт кроликов.

<html>

<head>

<meta charset=”utf-8”> *//подключаем русский язык*

<script>

function countRabbits(){ *// вызываемая ниже функция*

for(var i=1; i<=3; i++){

alert(“Кролик номер ” + i);

}

}

</script>

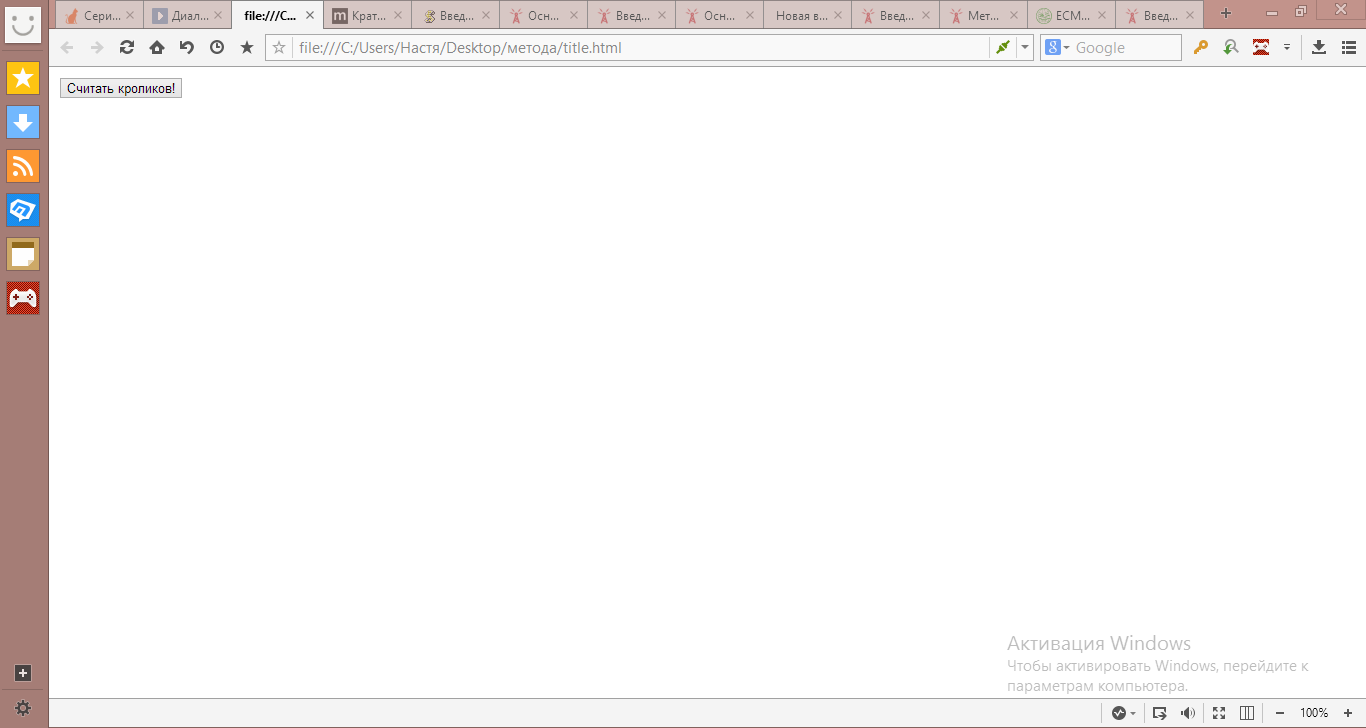
</head>

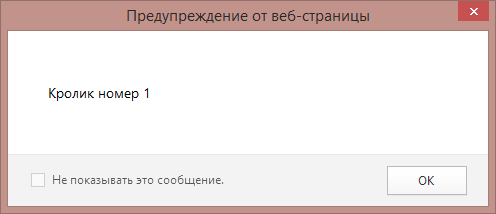
<body>

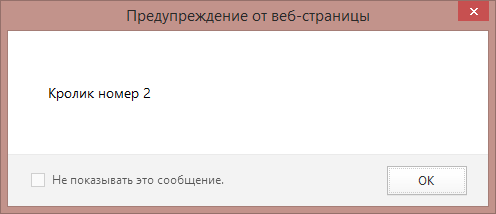
<input type=”button” onclick=”countRabbits()” value=”Считать кроликов!”/> *// вызываем функцию кликом по кнопке*

</body>

</html>







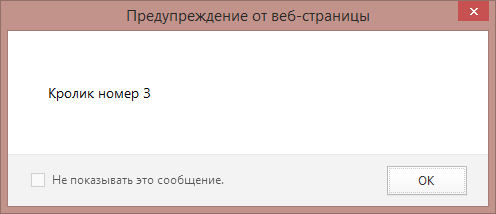


Рис.4. Результат работы программы.

# **Отладка JavaScript**

В *большинстве* программ для разработчиков самым лучшим другом программиста будет **консоль**. Многоцелевая панель используется для журналирования сообщений об ошибках, проверки DOM, отладки кода JavaScript и множества других задач. В зависимости от браузера консоль вызывается разными командами (кроме прямого выбора через меню):

* в браузере Chrome и Dragonfly для Opera – **Ctrl + Shift + I**
* Firebug - **F12**

Консоль автоматически выводит ошибки в коде, которые выявляются в ходе выполнения скрипта. Файл и строка указываются рядом с ошибкой, а нажатие клавиши мыши на ошибке перемещает фокус ввода в соответствующее ей место.

Консоль может не только показывать ошибки в коде скрипта. С помощью [Console API](http://getfirebug.com/wiki/index.php/Console_API) и [Command Line API](http://getfirebug.com/wiki/index.php/Command_Line_API) можно управлять выводом данных в консоль.  Самая известная и полезная команда **.log().**

При разработке кода формы очень полезно знать значения переменных, чтобы проверять правильность работы кода. Обычной практикой является использование функции **alert()** для визуального контроля. Однако, использование такого метода блокирует выполнение остальной части кода до нажатия кнопки в окне диалога.

Современным решением является использование метода **console.log**, который выводит значения переменных на панель консоли:

*сonsole.log(“Captain’s Log”); //выводит “Captain’s Log” в панель консоли*

Методможно использовать для вывода вычисленных значений:

|  |  |
| --- | --- |
|  | function calcPhotos() { |
|  | total\_photos\_diff = total\_photos - prev\_total\_photos; | |

|  |  |  |
| --- | --- | --- |
|  | // Выводим значения переменных в консоль | |
|  | console.log(total\_photos\_diff); |

|  |  |
| --- | --- |
|  | } |

Преимуществом такого подхода по сравнению с методом использования диалога **alert()** является то, что выполнение кода не прерывается, и разработчик может несколько раз выводить значения переменных для наблюдения за изменениями данных в реальном времени.

|  |  |
| --- | --- |
|  | var t = 3, |
|  | p = 1; | |

|  |  |
| --- | --- |
|  | function calcPhotos(total\_photos, prev\_total\_photos) { |
|  | var total\_photos\_diff   = total\_photos - prev\_total\_photos; | |

|  |  |
| --- | --- |
|  | // Выводим значения в консоль |

|  |  |
| --- | --- |
|  | console.log(total\_photos\_diff); |

|  |  |  |
| --- | --- | --- |
|  | // Обновляем значения | |
|  | t = t\*1.3; |

|  |  |
| --- | --- |
|  | p = p\*1.1; |

|  |  |
| --- | --- |
|  | setInterval(function() { |
|  | calcPhotos(t,p); |

|  |  |
| --- | --- |
|  | },100); |

***Прерывание хода выполнения скрипта***

Вывод информации в консоль является полезным средством, но код может выполняться очень быстро и при этом отслеживать много переменных.

Чтобы облегчить процесс отладки можно прерывать ход выполнения кода в определенной точке с получением доступа к данным. Для этого используются точки прерывания.

### *Работа с точками прерывания*

Для установки точки прерывания нужно перейти на закладку '*Scripts*' и выбрать нужный скрипт из списка. Теперь ищем строку, где нужно прервать ход выполнения скрипта, и жмем на поле с номером строки для активации - появится визуальный индикатор. Теперь перегружаем страницу и выполнение кода прервется в заданной точке:

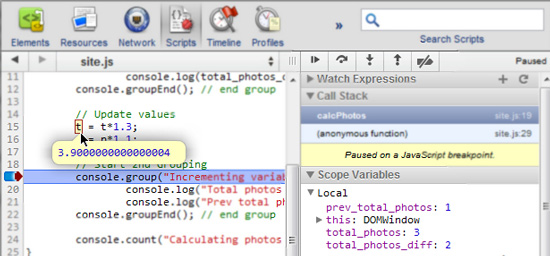


Рис.5. Работа с точками прерывания.

Когда выполнение прервется, можно поместить курсор мыши над любой переменной и отладчик выведет подсказку со значением в текущий момент.

Затем можно продолжить выполнение кода с помощью специальных кнопок, которые располагаются вверху боковой панели:

* http://ruseller.com/lessons/les1408/img_7.jpg “Continue”: продолжает выполнение кода до следующей точки прерывания.

* http://ruseller.com/lessons/les1408/img_8.jpg “Step Over”: выполняет следующую строку кода. Если код вызывает другую функцию, то отладчик не будет "погружаться" в ее код.

* http://ruseller.com/lessons/les1408/img_9.jpg “Step Into”: похоже на "Step over", за исключение того, что при вызове функции, отладчик переходит к первой строке внутри кода функции.

* http://ruseller.com/lessons/les1408/img_10.jpg "Step Out": если вы вошли в код функции с помощью кнопки "Step Into", то нажатие кнопки "Step out" вызовет выполнение кода функции до конца и переход к родительской функции.

На боковой панели можно отслеживать изменение состояния кода, включая динамику локальных и глобальных переменных.

### *Условные точки прерывания*

В процессе отладки кода иногда требуется останавливать выполнение кода только при соблюдении определенных условий. Например, если в вашем скрипте есть цикл, каждая итерация которого совершается за 50 миллисекунд, то будет очень неудобно запускать процесс выполнения после остановки на каждом шаге, когда нам нужна лишь 300 итерация. Для таких случаев есть условные прерывания.

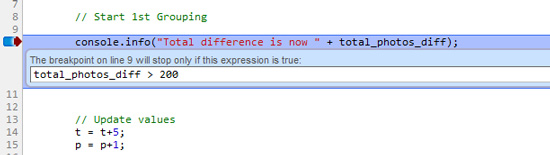


Рис.6. Условные точки прерывания.

В примере на рисунке выполнение кода не будет прерываться до тех пор, пока значение перменной **total\_photos\_diff** не станет больше 200.

Для активации условного прерывания нужно нажать правую клавишу мыши на точке прерывания и выбрать пункт 'Edit Breakpoint' для вывода диалога редактирования условий формирования прерывания.

### *Установка точки прерывания в коде*

Не всегда удобно устанавливать точки прерывания с помощью интерфейса инструмента разработчика в браузере. Иногда проще запустить отладчик из кода специальной командой. В примере, приведённом ниже, показано, как вы можете прервать выполнение кода при соблюдении определенных условий:

|  |  |
| --- | --- |
| 1 | if (total\_photos\_diff > 300) { |
| 2 | debugger;     // запускаем отладчик и прерываем выполнение кода | |

|  |  |
| --- | --- |
| 3 | } |

 Не бойтесь писать debugger в вашем коде — ошибки это нигде не вызовет.

**Пример 5.3.**

Разработка онлайн калькулятора при помощи языков HTML, JavaScript, CSS.

**HTML:**

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" //предназначен для указания типа текущего документа, т.к. HTML существует в нескольких вариантах(в нашем случае означает : “Строгий синтаксис HTML”)

<html>

<head>

<meta http-equiv="Content-Type" content="text/html" charset="utf-8" />

<meta HTTP-EQUIV="Content-Language" CONTENT="ru">//русский текст

<link rel="stylesheet" media="screen, print, handheld" type="text/css" href="style.css" />//подключаем наш документ css, содержащий стиль нашего калькулятора

<script type="text/javascript" src="script.js"></script>//подключаем документ, содержащий наш скрипт

<title>Пример лабораторной работы 5</title>

</head>

<body>

<table class="calculator" id="calc">//начинаем построение нашей таблицы

<tr>

<td colspan="4" class="calc\_td\_result">

<input type="text" readonly="readonly" name="calc\_result" id="calc\_result" class="calc\_result" onkeydown="javascript:key\_detect\_calc('calc',event);" />//создаём текстовое поле

</td>

</tr>

<tr>

<td class="calc\_td\_btn">

<input type="button" class="calc\_btn" value="CE" onclick="javascript:f\_calc('calc','ce');" />//создаём кнопку CE

</td>

<td class="calc\_td\_btn">

<input type="button" class="calc\_btn" value="&larr;" onclick="javascript:f\_calc('calc','nbs');" />//создаём кнопку nbs

</td>

<td class="calc\_td\_btn">

<input type="button" class="calc\_btn" value="%" onclick="javascript:f\_calc('calc','%');" />//создаём кнопку %

</td>

<td class="calc\_td\_btn">

<input type="button" class="calc\_btn" value="+" onclick="javascript:f\_calc('calc','+');" />//создаём кнопку +

</td>

</tr>

<tr>

<td class="calc\_td\_btn">

<input type="button" class="calc\_btn" value="7" onclick="javascript:add\_calc('calc',7);" />//создаём кнопку 7

</td>

<td class="calc\_td\_btn">

<input type="button" class="calc\_btn" value="8" onclick="javascript:add\_calc('calc',8);" />//создаём кнопку 8

</td>

<td class="calc\_td\_btn">

<input type="button" class="calc\_btn" value="9" onclick="javascript:add\_calc('calc',9);" />//создаём кнопку 9

</td>

<td class="calc\_td\_btn">

<input type="button" class="calc\_btn" value="-" onclick="javascript:f\_calc('calc','-');" />//создаём кнопку -

</td>

</tr>

<tr>

<td class="calc\_td\_btn">

<input type="button" class="calc\_btn" value="4" onclick="javascript:add\_calc('calc',4);" />//создаём кнопку 4

</td>

<td class="calc\_td\_btn">

<input type="button" class="calc\_btn" value="5" onclick="javascript:add\_calc('calc',5);" />//создаём кнопку 5

</td>

<td class="calc\_td\_btn">

<input type="button" class="calc\_btn" value="6" onclick="javascript:add\_calc('calc',6);" />//создаём кнопку 6

</td>

<td class="calc\_td\_btn">

<input type="button" class="calc\_btn" value="x" onclick="javascript:f\_calc('calc','\*');" />//создаём кнопку x

</td>

</tr>

<tr>

<td class="calc\_td\_btn">

<input type="button" class="calc\_btn" value="1" onclick="javascript:add\_calc('calc',1);" />//создаём кнопку 1

</td>

<td class="calc\_td\_btn">

<input type="button" class="calc\_btn" value="2" onclick="javascript:add\_calc('calc',2);" />//создаём кнопку 2

</td>

<td class="calc\_td\_btn">

<input type="button" class="calc\_btn" value="3" onclick="javascript:add\_calc('calc',3);" />//создаём кнопку 3

</td>

<td class="calc\_td\_btn">

<input type="button" class="calc\_btn" value="&divide;" onclick="javascript:f\_calc('calc','');" />//создаём кнопку “

</td>

</tr>

<tr>

<td class="calc\_td\_btn">

<input type="button" class="calc\_btn" value="0" onclick="javascript:add\_calc('calc',0);" />//создаём кнопку 0

</td>

<td class="calc\_td\_btn">

<input type="button" class="calc\_btn" value="&plusmn;" onclick="javascript:f\_calc('calc','+-');" />//создаём кнопку +\-

</td>

<td class="calc\_td\_btn">

<input type="button" class="calc\_btn" value="," onclick="javascript:add\_calc('calc','.');" />//создаём кнопку ‘,’

</td>

<td class="calc\_td\_btn">

<input type="button" class="calc\_btn" value="=" onclick="javascript:f\_calc('calc','=');" />//создаём кнопку =

</td>

</tr>

</table>

<script type="text/javascript">

document.getElementById('calc').onload=init\_calc('calc');

</script>

</body>

</html>

**JavaScript:**

calc\_array = new Array();//объявляем массив

var calcul=0;//объявляем переменную

var pas\_ch=0;

function $id(id)

{

return document.getElementById(id);

}

function f\_calc(id,n)//создаём функцию f\_calc с параметром n типа id

{

if(n=='ce')

{

init\_calc(id);

}

else if(n=='=')

{

if(calc\_array[id][0]!='=' && calc\_array[id][1]!=1)

{

eval('calcul='+calc\_array[id][2]+calc\_array[id][0]+calc\_array[id][3]+';');

calc\_array[id][0] = '=';

$id(id+'\_result').value=calcul;

calc\_array[id][2]=calcul;

calc\_array[id][3]=0;

}

}

else if(n=='+-')

{

$id(id+'\_result').value=$id(id+'\_result').value\*(-1);

if(calc\_array[id][0]=='=')

{

calc\_array[id][2] = $id(id+'\_result').value;

calc\_array[id][3] = 0;

}

else

{

calc\_array[id][3] = $id(id+'\_result').value;

}

pas\_ch = 1;

}

else if(n=='nbs')

{

if($id(id+'\_result').value<10 && $id(id+'\_result').value>-10)

{

$id(id+'\_result').value=0;

}

else

{

$id(id+'\_result').value=$id(id+'\_result').value.slice(0,$id(id+'\_result').value.length-1);

}

if(calc\_array[id][0]=='=')

{

calc\_array[id][2] = $id(id+'\_result').value;

calc\_array[id][3] = 0;

}

else

{

calc\_array[id][3] = $id(id+'\_result').value;

}

}

else

{

if(calc\_array[id][0]!='=' && calc\_array[id][1]!=1)

{

eval('calcul='+calc\_array[id][2]+calc\_array[id][0]+calc\_array[id][3]+';');

$id(id+'\_result').value=calcul;

calc\_array[id][2]=calcul;

calc\_array[id][3]=0;

}

calc\_array[id][0] = n;

}

if(pas\_ch==0)

{

calc\_array[id][1] = 1;

}

else

{

pas\_ch=0;

}

document.getElementById(id+'\_result').focus();

return true;

}

function add\_calc(id,n)//создаём функцию add\_calc

{

if(calc\_array[id][1]==1)

{

$id(id+'\_result').value=n;

}

else

{

$id(id+'\_result').value+=n;

}

if(calc\_array[id][0]=='=')

{

calc\_array[id][2] = $id(id+'\_result').value;

calc\_array[id][3] = 0;

}

else

{

calc\_array[id][3] = $id(id+'\_result').value;

}

calc\_array[id][1] = 0;

document.getElementById(id+'\_result').focus();

return true;

}

function init\_calc(id)//создаём функцию init\_calc

{

$id(id+'\_result').value=0;

calc\_array[id] = new Array('=',1,'0','0',0);

document.getElementById(id+'\_result').focus();

return true;

}

function key\_detect\_calc(id,evt)//создаём функцию key\_detect\_calc

{

if((evt.keyCode>95) && (evt.keyCode<106))

{

var nbr = evt.keyCode-96;

add\_calc(id,nbr);

}

else if((evt.keyCode>47) && (evt.keyCode<58))

{

var nbr = evt.keyCode-48;

add\_calc(id,nbr);

}

else if(evt.keyCode==107)

{

f\_calc(id,'+');

}

else if(evt.keyCode==109)

{

f\_calc(id,'-');

}

else if(evt.keyCode==106)

{

f\_calc(id,'\*');

}

else if(evt.keyCode==111)

{

f\_calc(id,'');

}

else if(evt.keyCode==110)

{

add\_calc(id,'.');

}

else if(evt.keyCode==190)

{

add\_calc(id,'.');

}

else if(evt.keyCode==188)

{

add\_calc(id,'.');

}

else if(evt.keyCode==13)

{

f\_calc(id,'=');

}

else if(evt.keyCode==46)

{

f\_calc(id,'ce');

}

else if(evt.keyCode==8)

{

f\_calc(id,'nbs');

}

else if(evt.keyCode==27)

{

f\_calc(id,'ce');

}

return true;

}

**CSS:**

.calculator

{

width:300px;

height:300px;

background-color:#eeeeee;

border:2px solid #CCCCCC;

margin:auto;

padding-left:5px;

padding-bottom:5px;

}

.calculator td

{

height:16.66%;

}

.calc\_td\_result

{

text-align:center;

}

.calc\_result

{

width:90%;

text-align:right;

}

.calc\_td\_calculs

{

text-align:center;

}

.calc\_calculs

{

width:90%;

text-align:left;

}

.calc\_td\_btn

{

width:25%;

height:100%;

}

.calc\_btn

{

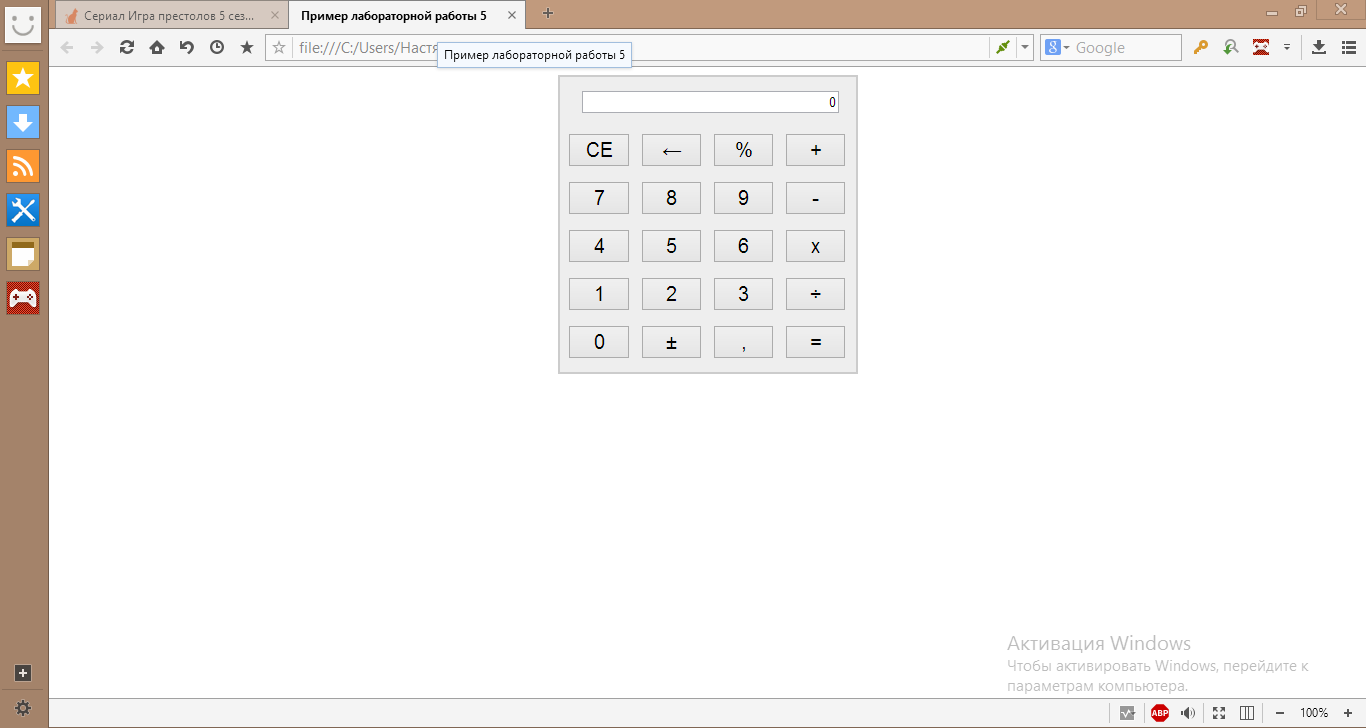
width:90%;

height:90%;

font-size:20px;

}

Результат выполнения программы:



# **Контрольные вопросы**

1. Что такое HTML?
2. Какие особенности языка HTML?
3. Основные элементы тега HEAD?
4. Что такое DHTML и чем он отличается от HTML?
5. Что такое JavaScript?
6. Разница между компиляцией и интерпретацией? Что использует язык JavaScript?
7. В чём уникальность JavaScript?
8. Каким образом осуществляется вставка JS кода в HTML?
9. Чем глобальные переменные отличаются от локальных?
10. Что такое “*Браузерное событие*”?
11. Что такое “*обработчик*” события?
12. Какие виды браузерных событий вам известны?

# **Задания для самостоятельного выполнения**

1. А. Создать бегущую строку.

Б. Создать надпись, всегда следующую за курсором мыши.

1. А. Реализовать автоматическую смену цветов фона.

Б. Психологический тест из 5-6 вопросов. Выбор варианта ответа осуществляется с помощью радио-кнопки, результаты обрабатываются автоматически и выдается результат.

1. А. Создать часы, отображающие текущее время.

Б. Реализовать автоматическое изменение размера текста.

1. А. Реализовать автоматическое изменение цвета текста.

Б. Создать выпадающий список с цветами, выбирая которые меняется цвет фона.

1. А. Реализовать отсчет количества дней до дня рождения студента.

Б. Создать изображение, всегда следующее за курсором мыши.

1. А. Отображать текущую дату.

Б. Создать счетчик посещений.

1. А. Создать бегущую строку с кнопкой остановки и запуска.

Б. Создать кнопку, меняющую цвет фона случайным образом.

1. А. Реализовать отсчет количества дней до нового года.

Б. Создать анкету из 5-6 пунктов и кнопку для очистки формы.

1. А. Создать изображение, движущееся по определенной траектории.

Б. Создать таблицу, сортировка полей которой должна осуществляться по алфавиту в прямом и обратном порядке.

1. А. Реализовать «салют» при нажатии на любую из клавиш мыши.

Б. Создать таблицу, сортировка полей которой должна осуществляться по номеру в прямом и обратном порядке.

1. А. При нажатии на кнопку должно выскакивать окно с текстом.

Б. Создать бегущую строку с кнопкой остановки и запуска.

1. А. Реализовать передвижение изображения с помощью мыши.

Б. Создать 3 кнопки, изменяющие фон, соответственно, в первой, второй и четвертой области.

1. А. Реализовать «змейку»: набираем предложение в поле ввода, нажимаем кнопку «пуск», далее, оно перемещается по экрану в различных направлениях (как по вертикали, так и по диагонали). Предусмотреть кнопки «пауза» и «стоп».

Б. Реализовать «Мастер гистограмм».