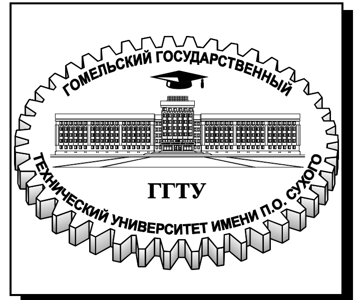


Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Институт повышения квалификации
и переподготовки кадров

Кафедра «Информатика»

А. И. Рябченко, А. А. Родионов, С. М. Горский

ПРОЕКТИРОВАНИЕ ДИНАМИЧЕСКИХ СТРАНИЦ

КУРС ЛЕКЦИЙ
по одноименной дисциплине
для слушателей специальности 1-40 01 74
«Web-дизайн и компьютерная графика»
заочной формы обучения

Гомель 2012

Библиотека ГГТУ им. П.О. Сухого

УДК 004.43(075.8)
ББК 32.973-018-1я73
Р98

Рекомендовано кафедрой «Информатика» ГГТУ им. П. О. Сухого
(*протокол № 12 от 06.06.2012 г.*)

Рецензенты: д-р техн. наук, проф. каф. «Информационные технологии» ГГТУ им. П. О. Сухого
И. А. Мурашко

Рябченко, А. И.

Р98 Проектирование динамических страниц : курс лекций по одноим. дисциплине для слушателей специальности 1-40 01 74 «Web-дизайн и компьютерная графика» заоч. формы обучения / А. И. Рябченко, А. А. Родионов, С. М. Горский. – Гомель : ГГТУ им. П. О. Сухого, 2012. – 144 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: http://alis.gstu.by/StartEK. – Загл. с титул. экрана.

Курс лекций ориентирован на формирование у слушателей ключевых компетенций, связанных с пониманием основных понятий и принципов построения динамических страниц для сети Интернет. В качестве языка реализации рассматривается JavaScript. Рассмотрены базовые средства JavaScript и методы их использования для решения конкретных задач, возникающих при разработке Web-страниц.

Изложена информация об истории создания языка и его связи с html, основных используемых типах данных и переменных, операторах языка, управляющих структурах. Рассмотрены основные вопросы по работе с функциями, массивами и объектами, строками и регулярными выражениями. Отдельное место в данном курсе занимают темы, связанные с объектной моделью web-документов, активно используемой на современном развитии интернет.

Для слушателей ИПК и ПК специальности 1-40 01 74 «Web-дизайн и компьютерная графика» заочной формы обучения.

УДК 004.43(075.8)
ББК 32.973-018-1я73

© Учреждение образования «Гомельский государственный технический университет имени П. О. Сухого», 2012

Библиотека ГГТУ им. П.О.Сухого

Содержание

РАЗДЕЛ 1. ИСТОРИЯ СОЗДАНИЯ JAVASCRIPT. JAVASCRIPT И HTML.

1.1. Знакомство с javascript
1.2. Восемь главных моментов при работе с javascript
1.3. Сферы использования javascript
1.4. Что можно сделать при помощи javascript
1.5. Совместная работа javascript и html

РАЗДЕЛ 2. ТИПЫ ДАННЫХ И ПЕРЕМЕННЫЕ

2.1. Ввод и вывод данных
2.2. Типы данных
2.3. Преобразование типов
2.4. Специальные символы
2.5. Переменные
2.6. Операторы
2.7. Комментарии

РАЗДЕЛ 3. ОПЕРАТОРЫ ЯЗЫКА JAVASCRIPT

3.1. Операции присваивания
3.2. Арифметические операции
3.3. Операции сравнения
3.4. Строковые операции
3.5. Условные операции
3.6. Операции со структурами данных
3.7. Приориет операций

РАЗДЕЛ 4. УПРАВЛЯЮЩИЕ СТРУКТУРЫ И ОРГАНИЗАЦИЯ ЦИКЛОВ

4.1. Условные операторы
4.2. Операторы организации циклов
4.3. Метки
4.4. Оператор with
4.5. Оператор switch

РАЗДЕЛ 5. ФУНКЦИИ, МАССИВЫ, ОБЪЕКТЫ

5.1. Функция как тип данных

Библиотека ГГТУ им.П.О.Сухого

5.2. Рекурсивные функции
5.3. Функция как переменная
5.4. Функция как объект
5.5. Массивы. Функции для работы с массивами
5.6. Объекты
5.7. Коллекции
5.8. Пользовательские объекты
РАЗДЕЛ 6. СТРОКИ И РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ
6.1. Объект string
6.2. Регулярные выражения
РАЗДЕЛ 7. ОБЪЕКТНАЯ МОДЕЛЬ ДОКУМЕНТА
7.1. Объект window
7.2. Объект document
РАЗДЕЛ 8. JAVASCRIPT И HTML-ФОРМЫ
8.1. Контейнер form
8.2. Текстовое поле ввода (объект text)
8.3. Списки вариантов (объекты select и option)
8.4. Кнопки
РАЗДЕЛ 9. ПРОГРАММИРОВАНИЕ ССЫЛОК
9.1. Объекты URL
9.1. События URL
РАЗДЕЛ 10. COOKIES. НЕВИДИМЫЙ КОД И БЕЗОПАСНОСТЬ
10.1. Понятие cookie
10.2. Невидимый код
10.3. Модель безопасности
РАЗДЕЛ 11.СОБЫТИЯ КЛАВИАТУРЫ И МЫШИ
11.1. События
11.2. События мыши
11.3. События клавиатуры
11.4. Свойства Клавиатурных Событий
РАЗДЕЛ 12. РАБОТА С ЭЛЕМЕНТАМИ DOM
12.1. Поиск элементов в dom
12.2. Свойства Dom-Элементов
12.3. Изменение страницы посредством dom

Библиотека ГГТУ им. П.О.Сухого

РАЗДЕЛ 1. ИСТОРИЯ СОЗДАНИЯ JAVASCRIPT. JAVASCRIPT И HTML.

1.1. ЗНАКОМСТВО С JAVASCRIPT

Как и все, что связано с Web, новая технология JavaScript гораздо моложе Java. **JavaScript** сначала был разработан компанией Netscape, поэтому его первое имя — **LiveScript**. Ожидалось, что этот язык сценариев расширит возмож ности HTML и выступит в качестве частичной альтернативы большому числу CGI-сценариев,

После выхода Java компания Netscape начала работать вместе с компанией Sun над созданием языка сценариев, чей синтаксис и семантика по плану должны были тесно связываться с Java. Отсюда возникло и название **JavaScript**. В результате совместных усилий Netscape и Sun и был создан этот язык.

Одна из причин, послуживших созданию JavaScript, была связана с потребностью в присутствии логики и интеллекта не только со стороны сервера, но и клиента. В этом случае все выполнение переходит к серверу, причем даже такие простые операции, как проверка допустимости данных.

Внедрение логики в браузер существенно усилило клиента и превратило отношение в истинную *клиент-серверную* систему.

С момента своего появления (*декабрь 1995 г.*) язык JavaScript имел существенную поддержку в лице ведущих производителей, в числе которых **Apple, Borland, Sybase, Informix, Oracle, Digital, HP и IBM.** JavaScript продолжает развиваться, внедряясь не только в современные браузеры, но и в приложения, созданные различными компаниями.

Понимая всю важность создания Web-сценариев, **Microsoft** решила поддержать JavaScript. Однако компания Netscape предпочла передать Microsoft лишь только лицензию на право использования технологии. На основе общедоступной документации в Microsoft был создан "перепроектированый" JavaScript — **JScript**, поддерживаемый Internet Explorer 3.0 и выше. JScript 1.0 плохо совместим с JavaScript 1.1, используемым Netscape Navigator 3.0 и более поздними браузерами.

Компании Netscape, Microsoft и другие производители решили подогнать язык под стандарты ECMA (European Computer Manufacturer's Association). В тот период ECMA составила спецификацию языка — ECMAScript, что поддержали все

Библиотека ГГТУ им.П.О.Сухого

производители. Хотя стандарты ECMA оказывают существенную помощь, все же компании Netscape и Microsoft продолжают развивать языки JavaScript и JScript, выходя при этом за пределы стандартов.

Помимо JScript, еще одним конкурентом JavaScript является **VBScript**, созданный с целью упрощения Web-разработок на Visual Basic (VB). Из-за отсутствия поддержки со стороны Netscape, VBScript используется для интрасетей (или Internet-сайтов). Он пользуются большой популярностью у пользователей Microsoft Internet Explorer.

1.2. ВОСЕМЬ ГЛАВНЫХ МОМЕНТОВ ПРИ РАБОТЕ С JAVASCRIPT

1. JavaScript можно внедрить в HTML

Программы JavaScript обычно заключены в HTML-документах, и выполняются в них же. Большинство объектов JavaScript представлены соответствующими дескрипторами HTML, поэтому код находится на уровне клиентской части языка. Чтобы стать настоящим разработчиком на JavaScript, следует досконально изучить HTML,

HTML используется в JavaScript как мостик к тому, что называется *средой разработки Web-приложений.* Возможности HTML расширяются за счет обеспечения дескрипторов HTML событиями и организации управляемой событиями среды выполнения.

2. JavaScript зависит от среды

JavaScript — язык сценариев, а не инструмент. После включения в документы HTML JavaScript начинает зависеть от браузера. Если же поддержка со стороны браузера отсутствует, тогда ваш код выполняться не будет. Более того, если пробле- ма поддержки браузером решена не будет, код просто высветится на странице как текст.

Как и большинство языков сценариев, JavaScript интерпретируется браузером. JavaScript не компилируется в бинарный код наподобие **.exe,** но все же остается частью документа HTML. Недостаток интерпретируемого языка для выполнения кода затрачивается много времени, поскольку браузер компилирует директивы во время выполнения. Однако есть и преимущество - можно легко и быстро усовершенствовать исходный код. Не переживайте по поводу старых версий сценариев JavaScript. Если вы замените их в исходном

Библиотека ГГТУ www.gstu.by

документе HTML, то новый код будет выполняться при каждом получении пользователем доступа к документу.

3. JavaScript слабо типизированный язык

JavaScript отличается от типизированных языков, таких как Java и C++, которые требуют объявления всех переменных определенных типов перед началом их использования. JavaScript, наоборот, очень гибок. Вы не обязаны объявлять переменные специального типа. Если специальный тип вам неизвестен, то можно работать с переменной. Хотя явное объявление переменных и отражает хороший тон программирования, вовсе не обязательно проделывать это.

4. Javascript -объектно-ориентированный язык

Netscape и другие компании отнесли JavaScript к объектно-ориентированным языкам (OOP — object-oriented programming). JavaScript действительно является *объектно-ориентированным языком.* Вы работаете с объектами, которые инкапсулируют данные (свойства) и поведение (методы). Объектная модель JavaScript основывается на экземплярах, а не на концепции наследования.

5. JavaScript язык, управляемый событиями

Большинство составленных кодов будет отвечать на события, генерируемые пользователем или системой. Язык JavaScript обладает поддержкой обработки событий. HTML-объекты, подобные кнопкам и текстовым полям, усовершенствованы с целью поддержки обработчиков событий. При переходе к JavaScript из Java или Visual Basic, эта управляемая событиями среда является второй натурой.

6. JavaScript это не Java

Как обсуждалось ранее, Java и JavaScript разрабатывались разными компаниями. Основная причина сходства имен кроется в маркетинговых соображениях. Во-первых, JavaScript тесно интегрирован в HTML, а Java-аплеты всего лишь связываются с HTML-документами через дескриптор **<applet>.** Сам же аплет хранится в другом файле, загружаемом из сервера.

Во-вторых, Java — более надежный и функционально полный язык, т.к. ему присущи свойства наподобие строгого контроля типов, объектной ориентации и наличию компилятора. Если Java

Библиотека БГУИР (watermark)

предназначен для создания аплетов и независимых приложений, то JavaScript — исключительно для написания сценариев.

Синтаксис JavaScript напоминает синтаксис Java. Если вы привыкните к управляющим структурам JavaScript, можете считать, что первый шаг к овладению Java уже сделан.

7. JavaScript многофункциональный язык

JavaScript — многоаспектный язык, который может применяться в различных контекстах для решения проблем, связанных с Web. Немного дальше рассматриваются сферы применения JavaScript. Основные цели, достигаемые при **помощи** JavaScript:

Усовершенствование и оживление статических HTML-страниц с помощью спецэффектов, анимации и баннеров.

- Проверка данных без передачи на сервер.
- Получение основы для разработки Web-приложений типа клиент-сервер.
- Разработка клиентских приложений.
- "Сращивание" со стороны клиента HTML-объектов, Java-аплетов, элементов управления ActiveX и подключаемых модулей Netscape.
- Получение расширения к Web-серверу.
- Моделирование связи с базами данных без использования CGI.

8. JavaScript развивающийся язык

Ранее в главе упоминалось, насколько современен **JavaScript** как технология. Учитывая резкие перемены в Web, несложно понять, что JavaScript продолжает свое развитие в качестве языка. При разработке приложений JavaScript требуется думать не только о том, поддерживает ли тот или иной браузер JavaScript, но еще и о том, какое именно подмножество языка. Не мудрено попасть в безвыходную ситуацию, поскольку количество версий перешло всякие мыслимые пределы: JavaScript — 6 версий (1.0—2.0), JScript — огромное количество версий **(1.0—5.8** и еще несколько версий наподобие x.x.x).

Библиотека ГГТУ им. П.О.Сухого

1.3. СФЕРЫ ИСПОЛЬЗОВАНИЯ JAVASCRIPT

Стоит еще раз напомнить: JavaScript — язык, а не инструмент. Поскольку это — язык сценариев, его можно применять в широком диапазоне приложений. Если рассматривать JavaScript в контексте разработки Web-страниц, то в большинстве своем применение касается клиентской части. На стороне сервера JavaScript применяется в средах Netscape Enterprise Server и Microsoft Active Server Pages.

Резкие изменения в Web-технологиях за последние годы привели к бурному росту Web. Перед обсуждением темы Web-приложений было бы полезно проследить эволюцию Web от простейшего "расширения" Interenet: до уровня "культуры", ставшей частью современной жизни. Эволюция условно делится на четыре фазы.

Фаза 1. Символьный гипертекст

Фаза 2. Графически-ориентированные статические HTML-документы

Фаза 3. Динамические HTML-документы

В течение первых двух фаз Web-страницы создавались при помощи текстового редактора HTML и помещались на Web-сервер. После размещения на сервере большинство страниц оставались статическими до тех пор, пока автор их не изменял. Статические страницы вполне подходят для некоторых целей, однако не для всех. Для динамической генерации HTML-документов Web-разработчики стали использовать CGI-сценарии (CGI-Common Gateway Interface, интерфейс общего шлюза), обеспечивая с их помощью создание HTML-документов "на лету". Как раз это и стало основой первого уровня взаимодействия с пользователем в Web. С таким усовершенствованием Web могла превратиться в платформу для гипертекстовых документов, и кроме того, служить средой прикладных приложений.

Фаза 4. Активные HTML-документы

Четвертая фаза развития Web началась в 1995 г. с появлением подключаемых модулей (plug-ins) в Netscape Navigator и существенно ускорилась за счет внедрения поддержки Java. Необходимо было усилить клиентскую сторону и отказаться от опоры исключительно на сервер при запуске приложений либо обработке информации, введенной пользователем.

Благодаря Java, Web более не является коллекцией документов HTML, но может быть истинной клиент-северной средой, в которой

Библиотека ГГТУ им. П.О.Сухого

клиент обладает определенной независимостью от сервера. JavaScript соответствует данным требованиям. В JavaScript, Java, ActiveX и других клиентских расширениях браузер становится мощнейшей операционной средой, в которой можно работать с Web-приложениями.

Использование Web в качестве среды проектирования — относительно новое явление. С появлением Java, JavaScript, ActiveX и других технологий, идея развития Web-приложений показалась весьма заманчивой.

Web как среда проектирования может выглядеть достаточно сложной. Учитывая распределенный характер Web, Web-приложения могут состоять из множества частей, в каждой из которых применяется так или иная технология.

Клиентская часть

Клиентская часть среды разработки Web-приложений состоит из 4-х строительных блоков:

- Браузеры
- HTML (Hypertext Markup Language, язык гипертекстовой разметки)
- Клиентские расширения(Java-апплеты, элементы управления ActiveX и подключаемые модули Netscape)
- Языки сценариев (JavaScript и JScript)

Клиентские расширения

По мере возрастания потребностей в активных Web-страницах, простое расширение браузеров уже не могло рассматриваться в качестве удобоваримого решения. Некоторые расширения были подключаемыми модулями независимых разработчиков, которые увеличивали эффективность браузеров. Однако появилась потребность работать с *выполняемым содержимым (executable content)* в самом браузере. Для самой технологии не обязательно привязываться к работе конкретного браузера, хотя он и поддерживает ее.

На сегодняшний день существуют три отдельных клиентских расширения. У них есть общие черты, равно как и отличия:

- Java-аплеты
- Элементы управления ActiveX
- Подключаемые модули Netscape

Библиотека ГГТУ им.П.О.Сухого

Клиентские языки сценариев

Последние штрихи магии клиентской части завершают клиентские языки сценариев. На сегодняшний день основным таким языком является JavaScript, тем не менее компания Microsoft выдвигает собственную альтернативу — VBScript. Наибольшей популярностью VBScript пользуется в кругу разработчиков на Visual Basic. В конце главы можно найти краткое сравнение возможностей JavaScript и VBScript.

Серверная часть

Серверная часть среды разработки Web-приложений состоит из собственно Web-сервера плюс расширения серверного программного обеспечения. Как будет показано несколько позже, эти расширения могут принимать различные формы и работать с различными технологиями.

Серверы. Web-сервер загружается после отправки со стороны клиента запроса на требуемые HTML-документы, после чего возвращает их для просмотра. Программное обеспечение сервера — это приложение, выполняющееся на машине с установленным протоколом TCP/IP. Среди наиболее популярных серверов находятся Netscape Enterprise Server **(NES),** Microsoft Internet Information Server **(IIS)** и Apache.

Серверные расширения. Собственно Web-сервер обеспечивает передачу статических **HTML-страниц** клиенту при запросе, а также множество других функций. С другой стороны, некоторые серверные расширения обеспечивают возможности, не реализованные в самом сервере. К таким возможностям относятся CGI, серверные API. JavaScript и Java.

Клиенты

Интерфейс общего шлюза (CGI, Common Gateway Interface**)** представляет собой стандарт де-факто для организации взаимодействия внешних программ с Web-серверами. Типовой сценарий заключается в генерации запроса из формы HTML и отправка его серверу. Запрос запускает CGI-программу или сценарий, размещенный в специальном каталоге на сервере. CGl-программа обрабатывает запрос и возврашает HTML-документ с результатами.

CGI-программа может разрабатываться на любом языке программирования: Java, C++, Visual Basic, PHP, т.е. на том, что

Библиотека ГГТУ им. П.О. Сухого

способно продуцировать код, могущий выполняться на Web-сервере. В мире Unix CGI-сценарии принято составлять на языках, подобных Perl или языкам командных оболочек Unix.

1.4. ЧТО МОЖНО СДЕЛАТЬ ПРИ ПОМОЩИ JAVASCRIPT?

После краткого знакомства с технологиями разработки Web-приложений самое время посмотреть на роль клиентской части JavaScript в Web-приложениях.

Клиентские приложения

JavaScript можно использовать для разработки полного клиентского приложения. Хотя JavaScript и не столь всеобъемлющий язык, каковым является Java, тем не менее, он проявляет существенные возможности при работе с дескрипторами HTML и связанными с ними объектами. Создание средних приложений с помощью Java сопряжено с гораздо большими сложностями, связанными с необходимостью взаимодействия с HTML, чем это есть в JavaScript. Очевидно, что в отдельных случаях JavaScript обеспечивает практически идеальную основу для разработки приложений.

1) Проверка допустимости данных

JavaScript обеспечивает для Web-разработчиков возможность выполнять проверку допустимости данных, вводимых пользователем, без необходимости обращения к серверу. Внутри кода JavaScript можно определить, являются ли значения, введенные пользователем, корректными, или, скажем, соответствуют ли они требуемому формату.

2) Создание интерактивных форм

JavaScript также используется для "оживления" форм HTML. Одна из задач оживления связана с проверкой достоверности данных. Другая предполагает реализацию дополнительных возможностей, недоступных в HTML, как то поддержка информации в строке состояния, открытие второго окна браузера для отображения, скажем, справочной информации и т.п.

3) Клиентские таблицы поиска

Помимо проверки достоверности данных, один из способов уменьшения потребности доступа к серверу заключается в

Библиотека ГГТУ им.П.О.Сухого

использовании JavaScript для генерации и поддержки клиентских таблиц поиска. Следует помнить, что в таком случае данные должны встраиваться в сам HTML-документ, поэтому использование таблиц поиска должно сводится к небольшим информационным базам данных, предназначенным только для чтения.

4) Поддержка состояния

В Web-среде, не поддерживающей концепцию состояния, JavaScript применяется для поддержки состояния при обмене между сервером и клиентом. Основной способ поддержки состояния связан с использованием наборов cookies (информации, сохраняемой браузером на стороне клиента). JavaScript обеспечивает как поиск, так и хранение cookie-наборов на клиентской части.

1.5. СОВМЕСТНАЯ РАБОТА JAVASCRIPT И HTML

HTML дает возможность создавать замечательные статические Web-страницы. **Хоть** эти документы зачастую и оказываются высокохудожественными, интересными и во многом полезными, JavaScript позволяет превратить статические Web-страницы в интерактивные и сделать их более чувствительными к пользовательским действиям и вводу. Расширение HTML-страниц с помощью JavaScript усиливает их возможности, а также существенно увеличивает гибкость кода.

JavaScript обладает рядом преимуществ перед формируемыми на сервере интерактивными документами при помощи **CGI,** поскольку написанные на JavaScript документы, вообще говоря, не зависят от обработки на сервере, а следовательно, быстрее реагируют на запросы пользователя и быстрее взаимодействуют с ним.

Структура HTML-документа

Структура HTML-документа зачастую очень важна. При неправильном размещении дескрипторов увеличивается вероятность возникновения нежелательных и неожиданных эффектов. Документ всегда должен начинаться с **<html>** и завершаться </html>. Два основных дескриптора внутри упомянутых — <head> и <body>. Внутри них можно размешать остальные дескрипторы.

Библиотека ГГТУ им. П.О. Сухого

Атрибуты, используемые в различных дескрипторах

class Определяет класс, к которому принадлежит HTML-дескриптор.

id Определяет некоторое уникальное значение для элемента при ссылке на полный документ.

name Присваивает имена экземпляру дескриптора. Это имя часто используется JavaScript для ссылок на конкретные
элементы либо дескрипторами <a> для организации ссылок в пределах одного документа.

style Определяет информацию о стиле.

Атрибуты дескриптора <script>

defer Это атрибут логического типа, который используется для уведомления браузера о том, генерирует ли
JavaScript-сценарий какое-либо содержимое.

language Отвергнутый в настоящее время, однако применяемый ранее атрибут для определения языка и версии,
используемых внутри дескрипторов.

src Этот атрибут определяет URL внешнего исходного JavaScript-файла.

type Атрибут, пришедший на замену language; он сообщает браузеру, какой язык используется внутри дескрипторов.

Внедрение JavaScript в HTML

JavaScript-сценарии интегрируются в HTML-документы с использованием пары дескрипторов **<script> и </script>.** HTML-документ может содержать множество подобных пар, причем каждая из них зачастую заключает в себе более одного набора операторов JavaScript. Для **<script>** обязательно требуется присутствие **</script>.** Атрибут **type** используется для определения языка, на котором реализован сценарий, а атрибут **src** — для задания имени внешнего файла, являющегося **источником** JavaScript-программы.

Атрибут **defer** — это простой атрибут, который сообщает браузеру, будет ли код, расположенный между открывающим и закрывающим дескрипторами **<script>,,** генерировать какое-либо содержимое. Другими словами, используется ли в коде метод **document.write().**

Библиотека ГГТУ им.П.О.Сухого

Ниже приводится пример использования данного атрибута:

```
<script type="text/javascript" defer>
        <!—
        // Просто объявим переменную; пока не
стоит
        // ее выводить на страницу
        var myVar = 500;
        //>
</script>
```

language

В самой последней версии языков HTML и XHTML атрибут **language** отвергнут. Традиционно он использовался для определения имени языка и версии JavaScript, которые использовались для кода, расположенного между парой **<script>.** Формат этого дескриптора выглядит так:

src

Существуют два пути интегрирования операторов **JavaScript** в HTML-документ. Выбор пути зависит от требований при просмотре и изменении кода. Первый вариант позволяет рассматривать все коды одновременно и предполагает написание операторов JavaScript непосредственно в **HTML-документе.** Все эти операторы встраиваются в HTML-страницу между парой дескрипторов **<script>;** они носят название *встроенных* ***(inline)*** сценариев. Второй вариант (доступный только в JavaScript 1.1 и старше) предполагает сохранение JavaScript-кода в отдельном файле с расширением js. В таком случае появляется возможность обращения к файлу в открывающем дескрипторе **<script>.**

Ниже показан первый вариант - встраивание JavaScript в HTML-страницу:

```
<script type="text/javascript"> function options () {
document.write("Встраивание кода”); }
</script>
```

А сейчас — вариант с размещением сценария во внешнем файле:

```
<script src="/JScripts/myscript.js" </script>
```

Во втором варианте потребуется создать файл с именем **myscript.js,** который будет содержать одну строку кода:

document.write ("Вызов из отдельного файла")

Библиотека БГТУ имени П.О.Сухого

В случае использования последнего варианта между парой дескрипторов **<script>** желательно поместить один оператор, тем самым обеспечивая пользователям возможность обратной связи на случай, если файл с расширением Доказывается некорректным или недоступным. Иначе пользователь будет лицезреть неправильное поведение Web-страницы, не понимая **его** причин.

При загрузке сценария из отдельного файла необходимость в указании атрибута **language** не возникает до тех пор, пока применяется файл с расширением .js.

Пользуясь этим методом, можно изменять JavaScript-код без какого-либо открытия и риска возникновения нежелательных изменений в HTML-страницах. Следовательно, код подобного рода представляется более модульным и переместимым, поскольку он размещается за пределами HTML-документа.

Отрицательная сторона метода состоит в том, что приходится модифицировать два набора кода, в зависимости от изменений в коде JavaScript. Например, если в JavaScript-коде изменяется имя функции, следует не забыть изменить соответствующее имя в обращениях к этой функции и в HTML-коде. Еще один минус связан с тем, что внешние JavaScript-файлы не могут содержать дескрипторы HTML; они должны содержать только операторы JavaScript.

type

Если просмотреть текущий стандарт HTML или предложенные рекомендации по XHTML, несложно заметить, что от атрибута **language** было решено отказаться, а вместо него использовать **type.** Значение, указываемое в этом новом атрибуте очень смахивает на указание в качестве содержимого внешнего исходного файла на JavaScript (js). Так, для JavaScript необходимо задать **type="text/javascript",** а для VBScript — **type="text/vbscript".**

Рекомендации HTML 4.01 говорят, что в часть **<head>** документа следует помещать дескриптор **<meta>,** который будет определять заданный по умолчанию язык, используемый во всех сценариях. Это выглядит так:

```
<meta http-equiv="Content-Script-Type" content="text/javascript">
```

Здесь можно заменить **text/javascript на text/vbscript** или любой другой язык, который планируется использовать.

Библиотека ГГТУ им. П.О. Сухого

Просмотр кода JavaScript

Поскольку код на JavaScript можно записывать непосредственно в коде HTML, просматривать и редактировать его легко. Вероятно, вам уже знаком пункт меню браузера Document Source (или Page Source), который обеспечивает просмотр исходного текста HTML-страницы (он обычно находится в меню View).

При просмотре исходного кода документа можно также исследовать JavaScript-код, включенный в документ, как показано на рис. 2.2. (Здесь, очевидно, как раз тот случай, когда операторы JavaScript вызываются из файла **.js** вместо помещения их непосредственно в документ. Все что можно здесь наблюдать — это обращение к файлу **.js** в дескрипторе **<script>.)** Для JavaScript не требуется специальное средство просмотра, и поскольку характер языка — интерпретирующий, а не компилирующий, код отображается в исходном тексте документа.

Создание JavaScript-кода

Основы создания JavaScript-кода весьма просты. Достаточно только создать основную HTML-страницу (или отредактировать существующую) и затем вставить дескрипторы <script> в разделы **<head>** или **<body>.** Функции и другие элементы, которые применяются ко всей Web-странице, лучшие помещать в раздел **<head>.** При наличии необходимости генерировать текст для Web-страницы при помощи JavaScript, это лучше выполнять в разделе **<body>.**

Выполнение сценариев

Выполнение JavaScript-сценариев начинается в разное время, в зависимости от того, как они написаны. Если сценарий влияет на содержимое Web-страницы, например, вызывает метод **document.write(),** он **выполяется** по мере своего появления. Кроме того, существует обработчик событий **onLoad,** который выполняется только после полной загрузки HTML-документа в браузер.

Если **JavaScript-сценарии** сохранены в отдельном файле, они анализируются во время загрузки Web-страницы и перед любыми действиями пользователя.

Все операторы JavaScript, содержащиеся в теле функции, интерпретируются, однако запуск функции не происходит до тех пор, пока она не будет вызвана каким-то JavaScript-событием. Операторы

JavaScript, находящиеся за пределами тела функции, выполняются после загрузки документа в браузер, но во время визуализации документа. Результат выполнения станет очевиден для пользователей уже при первом просмотре Web-страницы.

Загрузка Web-страницы

Операторы JavaScript, которые требуют немедленной обработки, выполнятся сразу после загрузки Web-страницы, но до ее отображения в браузере. Функция **onLoad** может рассматриваться как исключение из этого правила.

Второй метод, в соответствие с которым выполняются операторы JavaScript, — это вызов функций. Любой оператор, содержащийся в функции, не будет выполняться до тех пор, пока JavaScript-событие не вызовет эту функцию. JavaScript-события генерируются на HTML-странице многими способами, включая действия пользователя и явные вызовы событий непосредственно изнутри сценария. Пользовательские действия в документе способны генерировать JavaScript-события в многих случаях, даже в самых неожиданных. Всегда следует тщательно протестировать все операторы JavaScript, дабы убедиться, что пользовательское взаимодействие с Web-страницей не вызывает нежелательные события.

Пользовательские действия или явные вызовы событий — это часто используемые методы выполнения JavaScript-кода. Исключительное преимущество JavaScript заключается в том, что он способен увеличивать количество взаимодействий пользователя с HTML-документом, поскольку обеспечивает способ немедленной обработки и оценки вводимой пользователем информации.

Учет браузеров, не поддерживающих JavaScript

Быстрые темпы изменений HTML и JavaScript требуют предельно осторожного отношения к браузерам, которые не поддерживают создаваемые документы. Не все браузеры поспевают за новыми изменениями в HTML, поэтому программисты вынуждены делать документы максимально дружественными по отношению ко всем браузерам и средам.

Необходимо тестировать HTML и JavaScript-страницы в как можно большем количестве браузеров; такая политика приведет к максимальной стабильности и применимости документов.

Библиотека ГГТУ им. П.О.Сухого

Хотя применение JavaScript делает усовершенствования HTML доступными для пользователей, необходимо всегда помнить, что старые браузеры не способны поддерживать JavaScript-код. Если окружить дескрипторами комментария HTML все операторы внутри пары <script> и </script>, более старые браузеры смогут отобразить Web-страницу, не показывая JavaScript-код. Пользователи не смогут получить полное впечатление от Web-страницы, но, по крайней мере, в браузере не появится нежелательный текст. Формат использования этих комментариев таков:

```
<script type="text/javascript">
<!-- скрыть код от старых
браузеров
// здесь начинается собственно
код
// а здесь код завершается
//-->
</script>
```

Написание кода

Как и в любом другом языке программирования, операторы JavaScript могут быть записываться в соответствие с различными методами. Очевидно, практика определения функций JavaScript в разделе **<head>** и последующий вызов этих функций в разделе **<body>** — самый эффективный способ использования преимуществ объектов JavaScript.

Сам по себе язык JavaScript не труден, и разработчики, владеющие технологией объектно-ориентированного программирования, не встретят слишком много препятствий. Как только будут схвачены основные концепции объектной методологии, создание функций JavaScript становится достаточно простым делом.

Несмотря на то что операторы HTML не чувствительны к регистру (хотя в предложенных рекомендациях XHTML подобное требование присутствует), операторы JavaScript — чувствительны. Не забывайте проверять регистр символов кода, чтобы гарантированно не иметь проблем из-за столь досадного, но все же распространенного недоразумения.

Приступая к написанию кода, следует помнить о следующем:

- Повторное использование кода
- Удобочитаемость

Библиотека ГГТУ им.П.О.Сухого

- Простота внесения изменений

Дескрипторы JavaScript можно использовать как в разделе **<body>, так** и в разделе **<head>** документа. Как упоминалось ранее, помещение дескриптора **<script>** в **<head>,** а не в **<body>** гарантирует, что все операторы будут просмотрены (и, в случае необходимости, выполнены) еще до того, как пользователь начнет взаимодействовать с документом. Не следует забывать и о том, что помещение операторов сценария в раздел **<body>** документа имеет массу опасных последствий.

Даже зная конкретный список дескрипторов и порядок построения документа, нельзя заранее гарантировать, будет ли пользователь взаимодействовать со сценарием предполагаемым способом или отреагирует на Web-страницу прежде, чем сценарий полностью загрузится или выполнится. В этом случае предполагаемый эффект не будет достигнут.

Удобочитаемость и простота модификации как непосредственно для вас, так и для последующих разработчиков, также существенно снизится, если JavaScript-операторы для функций не поместить в раздел **<head>.**

Бесконечные поиски кода, который на самом деле **отыскивается** легко, в конце-концов выведут из душевного **равновесия** даже самого терпеливого.

Стили столь же важны в JavaScript, как и в любом другом **языке** программирования. Сохранение строгости стиля, своевременное определение переменных и аккуратность форматирования в будущем сократят время разработки документов.

Создавая сценарий JavaScript, функцию за функцией, кусочек за кусочком, можно сформировать устойчивые интерактивные документы, которые будут обладать большими функциональными возможностями. Поскольку **JavaScript** интерпретируется, но не компилируется, процесс отладки не всегда полностью прозрачен.

РАЗДЕЛ 2. ТИПЫ ДАННЫХ И ПЕРЕМЕННЫЕ

2.1. ВВОД И ВЫВОД ДАННЫХ

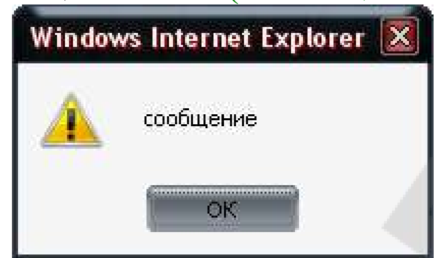
ВJavaScript предусмотрены довольно скудные средствадля ввода и выводадан-ных. Это вполне оправданно, поскольку JavaScript создавался в первую очередь как язык сценариев для веб-страниц.

Библиотека БГУИР (watermark)

Можно воспользоваться тремя стандартными методами для ввода и вывода данных: alert(), prompt(), confirm()

alert

Данный метод позволяет выводить диалоговое окно с заданным сообщением и кнопкой ОК. Синтаксис соответствующего выражения имеет следующий вид: общение alert("сообщение")



Вообще говоря, сообщение представляет собой данные любого типа: последовательность символов, заключенную в кавычки, число (в кавычках или без них), переменную или выражение.

confirm

Метод confirm позволяет вывести диалоговое окно с сообщением и двумя кнопками — ОК и Отмена (Cancel). В отличие от метода alerl этот метод возвращает логическую величину, значение которой зависит от того, на какой из двух кнопок щелкнул пользователь. Если он щелкнул на кнопке ОК. то возвращается значение true (истина, да);еслижеон щелкнул на кнопке Отмена, то возвращается значение false (ложь, нет). Синтаксис применения метода confirm имеет следующий вид:

confirm(сообщение)

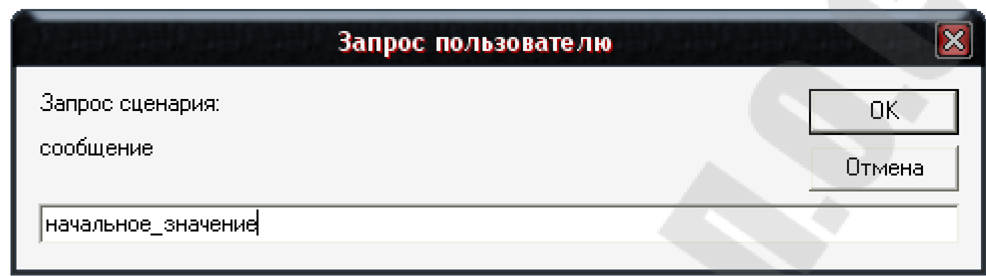
prompt

Метод prompt позволяет вывести на экран диалоговое окно с сообщением, а также с текстовым полем, в которое пользователь может ввести данные. Кроме того, в этом окне предусмотрены две кнопки: ОК и Отмена (Cancel) и данный метод принимает два

Библиотека БГТУ им. П.О. Сухого

параметра: сообщение и значение, которое должно появиться в текстовом поле ввода данных по умолчанию. Если пользователь щелкнет на кнопке ОК, то метод вернет содержимое поля ввода данных, аесли он щелкнетна кнопке Отмена, то возвращается логическое значение **false** (ложь, нет).

Синтаксис prompt(сообщение,начальное_значение)



2.2. ТИПЫ ДАННЫХ

В любом языке программирования очень важно понятие типа данных. Если не осознать его с самого начала, то потом придется часто сталкиваться со странным поведением созданной вами программы.

Типы данных в JavaScript
Строковьй или символьный (siring)
Числовой (number)
Логический (булевский, Boolean)
Null
Объект (object)
Функция (function)

Заметим, что строка "не содержащая ни одного символа (даже пробела), называется пустой. При этом строка, содержащая хотя бы один пробел, не пуста.

Данные логического типа могут иметь одно из двух значений: **true** или **false**. Эти значения записываются без кавычек. Значение true означает истину (да), а false -ложь (нет).

При создании программ на JavaScript за типами данных следит сам программист. Если он перепутает типы, то интерпретатор не зафиксирует ошибки, а попытается привести данные к нужному типу, чтобы выполнить указанную операцию.

Библиотека БГТУ им. П.О. Сухого

Например, результатом вычисления выражения 2+3 будет число 5, а выражения 2+"3"— строка "23", состоящая из двух цифровых символов.

2.3. ПРЕОБРАЗОВАНИЕ ТИПОВ

Для преобразования строк в числа в JavaScript предусмотрены встроенные функции parseInt() и parseFloat().

функция parseInt(строка, основание) преобразует указанную в параметре строку в целое число в системе счисления по указанному основанию (8, 10 или 16). Если основание не указано, то предполагается 10, то есть десятеричная система счисления.

Примеры

parseInt<"3. 14") // результат= 3
parseInt("-7. 87S-) ⁄, результат = -7
parseInt("435") // результат = 435
parseInt("Вася") // результат = NaN, го есть не является числом
parseInt('15', 8) // результат = 13
parseInt("0xFF", 16) // результат = 255

Обратите внимание, что при преобразовании в целое число округления не происходит: дробная часть просто отбрасывается.

Функция parseFloat(строка) преобразует указанную строку в число с плавающей разделительной (десятичной, основание) точкой.

Примеры

parseFloat("3. l4") // результат= 3. 14
parseFloat("-7. 875") // результат = -7. 875
parseFloat("435") // результат = 435
parseFloat("Вася")// результат = NaN, то есть не является числом
parseFloat("17. 5") // результат = 435

Задача преобразования чисел в строки возникает реже, чем обратное преобразование. Чтобы преобразовать число в строку, достаточно к пустой строке прибавить это число, то есть воспользоваться оператором сложения. Например, вычисление выражения ""+3.14 даст в результате строку "3.14".

Библиотека ГГТУ им. П.О. Сухого

Для определения того, является л и значение выражения числом, ел ужит встроенная функция isNaN(строка). Вычисление этой функции дает результат логического типа. Если указанное значение не является числом, функция возвращает true, иначе — false. Однако здесь понятие «число» не совпадаете понятием «значение числового типа».

Примеры

isNaN(123) // результат false (то есть это - число)
isNaN(“123”) // результат false (то есть это - число,
хотя и в виде строки) "/
isNaN(“50 рублей”) // результат true (то есть это - не число)
isNaN(true) // результат false
isNaN(false) // результат false
isNaN(“Вася”) // результат true (то есть это - не число)

2.4. СПЕЦИАЛЬНЫЕ СИМВОЛЫ

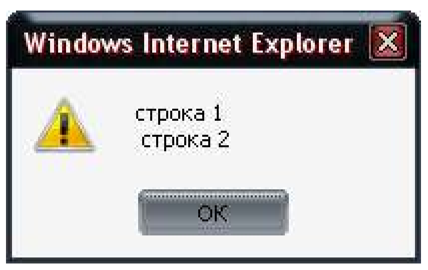
Иногда в сценарии может потребоваться использовать специальный знак либо их комбинацию, например, снмвол табуляции или новой строки. В этом случае управляющий код должен предваряться символом наклонной черты влево (\), как показано ниже:

\b — забой
\f — перевод страницы
\n — новая строка
\r — возврат каретки
\t — табуляция
\\ - наклонная черта влево
\' — одинарная кавычка
\" — двойная кавычка

Эти символы обычно используются при формировании строковых данных для их последующего отображения. Если требуется сэмулировать нажатие клавиши табуляции для выравнивания, скажем, двух столбцов данных, следует использовать знак табуляции (\t). Еще один пример – символ /r.

alert('строка 1\r строка 2')

Библиотека ГГТУ им.П.О.Сухого



Иногда требуется отобразить символы, имеющие служебное назначение. Как, например, отобразить кавычки, если они используются для задания строки символов? Для этой цели используется \" (обратная косая черта). Например, чтобы отобразить строку Акционерное общество "Рога и копыта" вместе с кавычками, сле-дуетнаписать такую строку: "Акционерное общество \"Рогаи копыта\"" Обратная косая черта указывает, что следующий непосредственно за ней символ не нужно интерпретировать как символ синтаксиса языка. Заметим, что эту же задачу можно решить и несколько иначе, используя кавычки различных видов (двойные и одинарные).

Во-первых, можно написагь так: 'Акционерное общество "Рога и копыта"' . в этом случае мы использовали одинарные кавычки в качестве признаков начала и конца всей строки.

Во-вторых, можно поменять местами кавычки различных видов: "Акционерное общество 'Рога и копыта'". Однако в этом случае название акционерного общества будет отображаться в одинарных кавычках.

Наконец, можновнешние кавычки оставить двойными, а внутренние одинарные кавычки продублировать: "Акционерное общество "Рога и копыта"". Тогда при отображении строки внутренние кавычки будут заменены на двойные.

Неправильное использование кавычек довольно часто вызывает проблемы у новичков.

Кавычки, обрамляющие строковые данные, должны быть одного вида и использоваться парами.

2.5. ПЕРЕМЕННЫЕ

Имена переменных

Имя переменной в JavaScript состоит из одной и более букв (A..Z и a..z), цифры (0..9) и символов подчеркивания (_). однако не может начинаться с цифры. При выборе имен переменных нельзя

Библиотека ГГТУ им. П.О.Сухого

использовать ключевые слова, то есть слова, используемые в определениях конструкций языка. Например, нельзя выбирать слова var, if, else, const, true, false, function, super, switch и ряд других. Следующих два имени JavaScript воспринимает как различные:

internetAddress

internetaddress

Вот еще корректные имена:

_lastName

n

number_2

Когда имя переменной по смыслу содержит одно слово, при ее записи принято использовать строчные буквы, а вот при наличии в имени смысловых двух или трех слов, то первое слово начинается со строчной, а все последующие — с прописных.

JavaScript является репклроэависнмычиыком. Это означает, что изменение регистра символов (с прописных на строчные и наоборот) в имени переменной приводит к другой переменной. Например: variable, Variable и vaRiabLe — различные переменные.

Создание переменных

Создать переменную в программе можно несколькими способами:

1) С помощью оператора присвоения значений в формате: имя_переменной= значение

Пример:

myName = "Иван"

2) С помощью ключевого слова var (от variable — переменная) в формате: var имя_переменной В этом случае созданная переменная не имеет никакого значения, но может его получить в дальнейшем с помощью оператора присвоения.

Приимер:

var myName;

myName ="Иван";

3) С помошью ключевого слова var и оператора присвоения в формате: var имя_переменной= значение

Пример:

var myName = "Иван"

В отличие от многих других языков программирования, при инициализации переменной не нужно описывать ее тип. Переменная может иметь значения различных типов и неоднократно их изменять.

Библиотека ГГТУ им. П.О.Сухого

Одно ключевое слово var можно использовать для инициализации сразу нескольких переменных, как с оператором присвоения, так и без него. При этом переменные и выражения с операторами присвоения разделяются запятыми, например:

var name = "Вася", address, x = 3. 14

Если x и y — две переменные, то выражение **x = y** интерпретируется так: переменной x присваивается значение переменной y. Иначе говоря, переменной можно присвоить значение другой переменной, указав справа от оператора ее имя.

Область действия переменных

Область действия (или видимости) переменных (scope) — это участки внутри программы, в которых возможна ссылка на переменную.

Локальные переменные

Переменная, объявляемая внутри функции является локальной. К значениям этой переменной может иметь доступ только данная функция. При каждом вызове функции все локальные переменные создаются, а после возврата функции — разрушаются. Если в другой функции присутствует объявление переменной с таким же именем, она рассматривается как совершенно другая переменная. Каждая из переменных адресует свой блок памяти.

Глобальные переменные

Если требуется, чтобы какая-то переменная совместно использовалась в более чем одной функции, следует объявить ее вне функций (но, конечно же, внутри пары

дескрипторов <script>). В этом случае данная переменная будет доступна в любой части всего приложения, в том числе и во всех функциях.

ПРИМЕЧАНИЕ: Рекомендуется объявлять глобальные переменные в разделе <head> HTML-страницы, тем самым гарантируя, что они загрузятся раньше любой другой части приложения.

2.6. ОПЕРАТОРЫ

Операторы предназначены для составления выражений. Оператор применяется к одному или двум данным, которые в этом случае

Библиотека ГГТУ им. П.О.Сухого

называются операндами. Например, оператор сложения применяется к двум операндам, а оператор логического отрицания — к одному операнду. Элементарное выражение, состоящее из операндов и оператора, вычисляется интерпретатором и, следовательно, имеет некоторое значение. В этом случае говорят, что оператор возвращает значение. Например, оператор сложения, примененный к числам 2 и 3, возвращает значение 5. Оператор имеет тип, совпадающий с типом возвращаемого им значения. Поскольку элементарное выражение с оператором и операндами возвращает значение, это выражение можно присвоить переменной.

2.7. КОММЕНТАРИИ

Начнем с операторов комментария. Они позволяют выделить фрагмент программы, который не выполняется интерпретатором, а служит лишь для пояснений содержания программы.

В JavaScript допустимы два вида операторов комментария:

// — одна строка символов, расположенная справа от этого оператора, считается комментарием;

/*   */ — все, что заключено между /* и */ считается комментарием; с помошью этого оператора можно выделить несколько строк в качестве комментария.

Не пренебрегайте комментариями в тексте программ. Это поможет при их отладке и сопровождении. На этапе разработки лучше сначала превратить ненужный фрагмент программы в комментарий, чем просто удалить (а вдруг его придется восстанавливать?),

РАЗДЕЛ 3. ОПЕРАТОРЫ ЯЗЫКА JAVASCRIPT

Для создания хороших программ необходимо научиться также оценивать и изменять данные, с которыми JavaScript-сценарии имеют дело. Инструментальные средства, реализующие подобную работу, носят название *операций.*

Операции — суть символы и идентификаторы, которые представляют способы изменения данных либо вычисления комбинаций выражений. Язык JavaScript поддерживает как бинарные,

Библиотека БГТУ им. П.О. Сухого

так и унарные операции. Бинарные операции работают с двумя операндами в выражении, например, 9 + x, тогда как для унарных операций достаточно только одного операнда, например, x++.

3.1. ОПЕРАЦИИ ПРИСВАИВАНИЯ

Рассмотрим операцию, о которой уже упоминалось ранее, — операцию присваивания. Ее основная функция заключается в присваивании переменной некоторого значения, помещая таким образом это значение в память.

Например, выражение x = 20 обеспечивает присваивание переменной x значения 20. Когда JavaScript встречает операцию **присваивания** (=), вначале анализируется правая часть с целью определения значения. После этого рассматривается левая часть — она должна представлять место для хранения значения. Если слева находится переменная, ей присваивается значение.

Операция присваивания всегда выполняется справа налево, так что выражение **20 = x** вызовет ошибку в JavaScript, поскольку будет предпринята попытка присвоить 20 новое значение.

JavaScript поддерживает 11 других операций присваивания, которые фактически являются комбинациями операции присваивания и арифметических или поразрядных операций. Сокращенные версии операций показаны ниже:

Комбинации операции присваивания и арифметических операций:

x += y сокращенная запись **для x = x + y**
x -= y сокращенная запись **для x = x - y**
x *= y сокращенная запись **для x = x * y**
x /= y сокращенная запись **для x = x / y**
x %= y сокращенная запись **для x = x % y**

Комбинации операции присваивания и поразрядных операций:

x <<= y сокращенная запись для **x = x << y**
x >>=y сокращенная запись для **x = x >> y**
x >>>= y сокращенная запись для **x = x >>> y**
x &= y сокращенная запись для **x = x & y**
x "= y сокращенная запись для **x = x ~ y**
x |= y сокращенная запись для **x = x | y**

3.2. АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

Библиотека ГГТУ им. П.О.Сухого

При работе с числами используются арифметические операции. К основным операциям этой группы относятся знак

"плюс" (+), суммирующий два значения, знак

"минус" (-), вычитающий одно значение из другого,

звездочка (*), перемножающая два значения,

косая черта (/), которая обозначает деление одного значения на другое.

Используя арифметические операции в комбинации с операцией присваивания, можно задавать новое значение переменной:

x = 7 + 9

Переменная x будет теперь равна 16, и ее можно использовать снова, в том числе присваивать ей новые значения:

x = x + 1

Увеличение значения переменной на 1 и затем присваивание ей же этого нового значения представляет собой достаточно стандартную операцию. Последняя настолько часто используется в компьютерных программах, что некоторые языки включают специальные со-кращенные операции, чтобы упростить запись увеличения и **уменьшения** значения переменной.

Для записи инкремента применяется ++, а для декремента --:

++i — то же самое, что и **i = i + 1** (похожим образом работает **i++**)

--i то же самое, что и **i = i - 1** (похожим образом работает **i--**)

Унарная операция отрицания (-) применяется в случаях, когда требуется заменить положительное значение на отрицательное и наоборот. Она названа унарной, т.к. работает только с одним операндом.

Операция взятия по модулю обозначается знаком процента (%). Найти модуль двух операндов означает найти остаток после деления первого операнда на второй. В примере **x = 10 % 3** переменная x получит значение 1, поскольку результатом деления 10 на 3 будет 3 с остатком 1. При помощи операции взятия по модулю можно легко определить, что одно число является множителем другого: при этом модуль этих двух чисел будет равен 0. Это истинно, например, для выражения x = 25 % 5. 25 разделить на 5 равно 5 без остатка, значит x будет равно 0.

3.3. ОПЕРАЦИИ СРАВНЕНИЯ

Библиотека ГГТУ им. П.О. Сухого

Операции сравнения используются именно для сравнения выражений. Выражения, которые используют операции сравнения, отвечают на вопрос, связанный с двумя значениями. Ответом может быть **true** или **false.**

Два знака "равно" (==) обозначают операцию равенства. Использовать операцию равенства между двумя операндами — означает определить, являются ли значения этих двух операндов равными. Необходимо убедиться, что при написании программы используется именно требуемая операция. Еще раз следует подчеркнуть, что операция равенства (==) проверяет, являются ли два значения равными, тогда как операция присваивания (=) устанавливает значение переменной. В случае использования не той операции интерпретатор **JavaScript** даст об этом знать.

==	Операция равенства. Возвращает **true,** если операнды равны между собой.
!=	Операция неравенства. Возвращает **true,** если операнды не равны между собой.
>	Операция "больше". Возвращает **true,** если значение левого операнда больше значения правого операнда.
>=	Операция "больше либо равно". Возвращает **true,** если значение левого операнда больше либо равно значения правого операнда.
<	Операция "меньше". Возвращает **true**, если значение левого операнда меньше значения правого операнда.
>=	Операция "меньше либо равно". Возвращает **true**, еспи значение левого операнда меньше либо равно значения правого операнда.

Операции сравнения обычно используются в **JavaScript** для принятия решений. Они помогают выбирать путь, по которому будет следовать сценарий

3.4. СТРОКОВЫЕ ОПЕРАЦИИ

Набор строковых операций, доступный в JavaScript, включает все операции сравнения и операцию конкатенации (+). При помощи операции конкатенации строки соединяются вместе в одну длинную строку

Библиотека ГГТУ имени П.О.Сухого

```
<html> <head>
<title>JavaScript Unleashed</title> </head> <body>
<script type="text/javascript">
// Объявление переменных
 var a = "www";
 var b = "company";
 var   c = "com";
 var sumOfParts;
 var addressl;
 var address2;
 document.write ("Part a is equal to \""+ a + "\".">;
 document.write ("<br>Part a is equal to \""+ b + "\".">;
 document.write ("<br>Part a is equal to \""+ c + "\".\n">;
 sumOf Parts = a + "." + b + "." + c;
 addressl = "WWW.COMPANY.COM";
 address2 = "www.company.com";
 // Отображение результатов
 document.write ("<br><br>Is sumOf Parts equal to " + address1 +"? ") ;
 document.write (sumOf Parts == address1);
 document.write("<br>Is sumOfParts equal to " + address2 + "? ") ;
 document.write (sumOf Parts == address2) ;
 document.write("<br>Is sumOfParts greater than " + addressl + "? ") ;
 document.write(sumOfParts + addressl);
 </script>
</body>
</html>
```

Условное выражение может применяться для возврата данных любого типа, например, number или boolean.

(условное выражение) ? операторы_1 : операторы_2

Например:

```
type_time = (hour >= 12) ? "PM" : "AM"
```

```
<html>
<head>
<script language ="JavaScript">
<!--
var today = new date();
```

```
var secs = today.getSeconds();
(secs >=0 && secs <= 30) ?
  document.write("<body text=white bgcolor=blue> Это написано белым на синем");
  document.write("<body text=red bgcolor=black> Это написано красным на черном");
//-->
</script>
</body>
</html>
```

3.5. УСЛОВНЫЕ ОПЕРАЦИИ

Операция typeof

Операция typeof возвращает тип данных, хранящихся в операнде в текущий момент времени. Результат выдвается в виде строки.

typeof 33 возвращает строку **"number",**

typeof "A String" возвращает строку **"string",**

typeof true возвращает строку **"boolean",**

typeof null возвращает строку **"object".**

&& Логическая операция И (конъюнкция). Возвращает true, если оба выражения expressioni и expression2 имеют

значения true. В противном случае возвращается false. Рассмотрим примеры:

(1 > 0) && (2 > 1) возвращает true.

(1 > 0) && (2 < 1) возвращает false.

|| Логическая операция ИЛИ (дизъюнкция). Возвращает true, если хотя бы одно из значений expression1 или expression2 равно true. Если ни одно из expressioni и expression2 не равно true, возвращается false. Рассмотрим примеры:

(1 > 0) || (2 < 1) возвращает true.

(1 < 0) || (2 < 1) возвращает false.

! Логическая операция НЕ (отрицание) —унарная операция, которая возвращает противоположное значение булева выражения.

Библиотека БГУИР

Если expression равно true, возвращается false, а если expression — false, возвращается true. Эта операция не изменяет значения выражения, поскольку работает подобно арифметической операции отрицания. Рассмотрим примеры:

! (1 > 0) возвращает false.

! (1 < 0) возвращает true.

3.6. ОПЕРАЦИИ СО СТРУКТУРАМИ ДАННЫХ

Операции со структурами данных — термин, который используется при классификации двух операций, необходимых для работы со структурами данных. Структуры данных — это концептуальные основы, применяемые для хранения одной или более базовых частей данных организованным способом. BJavaScript таковыми структурами являются объекты, группирующие данные для конкретных целей.

Операция, уже знакомая тем, кто имел дело с объектами, обычно называется "точка". Обозначается она точкой (.) и носит название *операцией взятия составляющей структуры.* Она позволяет обращаться к переменной, функции или целому объекту, принадлежащему определенному объекту. Синтаксис операции таков:

objectName.variableName

objectName.functionName()

objectName.anotherObject

Часть данных, к которой обращаются, должна стоять справа от точки. Такой способ обращения к члену структуры (к переменной, функции или объекту) обычно называется "точечной нотацией".

Операция индексирования применяется при обращении к части данных из массива. Обозначается она парой квадратных скобок и позволяет получить доступ к любому элементу массива:

arrayName[indexNumber]

Операция индексирования предполагает использование целого числа (индекса элемента массива) **indexNumber. IndexNumber** определяет индекс в **arrayName** и обеспечивает доступ к любому члену массива.

3.7. ПРИОРИЕТ ОПЕРАЦИЙ

Библиотека ГГТУ им. П.О. Сухого

При создании выражений, которые включают более одной операции, необходимо помнить, что JavaScript не обязательно вычисляет выражение справа налево или наоборот. Каждая часть выражения вычисляется в по- рядке, основанном на приоритете каждой операции. Рассмотрим такой пример: x = a * b + c. Если вначале необходимо выполнить сложение, выражение заключа- ют в круглые скобки: x = a * (Ь + c) Круглые скобки — это операция, которая увеличи- вает приоритет заключенного в них выражения. Однотипные операции вычисляются слева направо.

Название операции	*Операция*
Запятая	,
Присваивание	+= -= *= /= <<= >>= >>>= &= *= !=
Условие	? :
Логическое ИЛИ	||
Логическое И	**&&**
Поразрядное ИЛИ	|
Поразрядное ИСКЛЮЧАЮЩЕЕ ИЛИ	^
Поразрядное И	**&**
Равенство	**== !=**
Сравнение	< <= >= >
Поразрядный сдвиг	<< >> >>>
Сложение/вычитание	+
Умножение/деление	* / **%**
Отрицание/приращение	! ++ --
Вызов/структурирование данных	( ) [ ]

Один из эффектов, связанных с приоритетами операций, заключается в определении типа данных результата выражения.

```
<html>
<head>
<title>Приоритеты операций</title>
</head>
<body>
<script type="text/javascript">
<!--
   var carLength = 2 + 3;
```

```
   document.write(carLength);
   carLength = "<br>" + 2 + 3 – "метров";
   document.write(carLength);
   carLength = "<br>Длина в метрах: " + 2 + 3;
   document.write(carLength);
   carLength = "<br>Длина в метрах: " + (2 + 3);
   document.writeln(carLength);
//-->
</script>
</body>
</html>
```

РАЗДЕЛ 4. УПРАВЛЯЮЩИЕ СТРУКТУРЫ И ОРГАНИЗАЦИЯ ЦИКЛОВ

Проектирование сценариев, предполагающих принятие решений во время выполнения, — наиболее интересная часть JavaScript. В сценарии, где требуется принимать решения на основе его текущего состояния, просто обеспечивают вывод вопроса и затем, в зависимости от **ответа,** выбирают путь.

4.1. УСЛОВНЫЕ ОПЕРАТОРЫ

Ранее рассматривалась операция ?:. Эта операция используется для создания **двухшагового** процесса. Сначала выражение оценивается на предмет равенства true или false. Далее, в зависимости от результата, возвращается одно из двух значений. Если выражение true, возвращается первое значение, а если — false — второе.

Встречаются сценарии, принимающие решение в зависимости от значения выражения, использующего операторы if и else. Однако результат будет другим. Вместо возвращения зависящего от результата значения, программа выбирает один из двух путей выполнения. Так, можно заставить JavaScript выполнять множество различных функций, зависящих от любой доступной информации.

Оператор if

Оператор if относится к числу наиболее популярных. Каждый язык программирования содержит его в той или иной форме, и его

Библиотека ГГТУ им. П.О.Сухого

использования избежать, как правило, не удается. Оператор if применяется следующим образом:

if (условие) {

[*оператора*] }

В качестве *условия* может указываться любое логическое выражение. Если результат *условия* равен **true,** выполняются *операторы,* посему работа программы продолжается. Если *условие* возвращает **false,** *операторы* игнорируются и работа продолжается.

```
<html>
<head>
Принятие решений при помощи оператора if</title>
</head>
<body>
<script type="text/javascript">
<!--
   // Объявление переменных
   var visitorlnterest;
   // visitorlnterest = "New Products";
   visitorlnterest = "Technical Support";
   document.writeln("Привет, меня зовут Виктор Франкенштейн!");
   // Оценка значения visitorlnterest
   if (visitorlnterest == "New Products") {
      document. writeln ("Спасибо за интерес, проявленный к товарам!
");
   }
   if (visitorlnterest == "Technical Support") {
      document.writeln("СЧейчас доступна и техническая поддержка.
");
      document.writeln("Однако вначале позвольте познакомить вас");
      document.writeln("с новой продукцией!");
   }
   document, writeln ("<br>Нama новая продукция удовлетворит все
ваши");
   document.writeln("потребности!");
// —>
</script>
</body>
</html>
```

Оператор if..else

Иногда одной конструкции if оказывается недостаточно. Зачастую требуется также зарезервировать набор операторов, которые будут выполняться в случае, когда условное выражение возвращает false. Это можно сделать, добавив еще один блок непосредственно после блока if:

```
If (условие) {
  [операторы]
} else {
  [операторы] }
```

Кроме того, имеется возможность скомбинировать часть else с другим оператором if. Использование такого метода позволяет оценить несколько разных возможных сценариев перед выполнением нужной операции. Изящество данного метода состоит в том, что в конце можно определить тот же сегментом else. Формат упомянутого типа оператора выглядит так:

```
if (условие) {
  оператора
} else if (условие) {
  операторы
) else {
  операторы }
```

```
<html>
<head>
<title>Принятие решений при помощи оператора if</title>
</head>
<body>
<script type="text/javascript">
<!--
  // Объявление переменных
  var purchaseAmount;
  purchaseAmount = 10.00;
  if (purchaseAmount > 500.00) {
    document.writeln("Спасибо за покупку!");
  } else {
    document.writeln("Спасибо, однако я уверен, что у меня есть");
    document.writeln("еще кое-что, от чего вы не сможете
отказаться.");
```


Библиотека ГГТУ им. П.О.Сухого

```
    }
\\-->
</script>
</body>
</html>
```

4.2. ОПЕРАТОРЫ ОРГАНИЗАЦИИ ЦИКЛОВ

Организация цикла внутри сценария преследует широкий спектр целей. Рассмотрим одно простое, но весьма распространенное применение цикла. Например, написание программы, отображающей числа от 0 до 9, — на первый взгляд, быстрая и простая задача. Просто записываются 10 команд для отображения каждого числа:

```
document.writeIn ("0");
document.writeln ("1");
...
document.writeln ("9");
```

Это легко сделать для чисел от 0 до 9, но что если потребуется сосчитать до 1000? Можно последовать тому же способу, но тогда программа потребует гораздо больше времени на написание и загрузку в браузер. Наилучший способ счета до 1000 или любого другого числа заключается в использовании той же самой команды отображения, но с переменной вместо литерала. В данном случае потребуется обеспечить только одну вещь — инкрементаровать переменную и повторять команду отображения. JavaScript располагает соответствующими инструментами.

Оператор for

Оператор for используется для организации цикла в сценарии. Перед детальным изучением оператора for на основе листинга 7.4, рассмотрим его обобщенный синтаксис:

```
for([инициализация]; [условие]; [итерация]){
        операторы
}
```

Три выражения, заключенные в квадратные скобки, не являются обязательными, тем не менее, если даже убрать одно из них, точка с запятой все же потребуется. Именно точка с запятой обеспечивает должное место для каждого выражения.

Библиотека БГУИР

Выражение инициализации обычно применяется в целях инициализации переменной и даже объявления, что конкретная переменная является счетчиком цикла. Выражение условия должно иметь значение **true** перед каждым выполнением операторов, заключенных в фигурные скобки. Наконец, выражение цикла, как правило, инкрементирует или декрементирует значение переменной, которая используется в качестве счетчика цикла.

```
<html>
<head>
   <script language ="JavaScript">
   <!--
   function testloop() {
      var String1 = '<hr align="center" width="' ;
      document.open();
      for (var size = 5; size <= 50; size += 5)
               document.writeln (String1 + size + '%">');
      document.close();
   }
   //-->
   </script>
</head>
<body>
   <form>
      <input type="button"
      value="Test the loop"
      onClick="testloop()">
   </form>
</body>
</html>
```

Библиотека ГГТУ им. П.О. Сухого

ПРИМЕЧАНИЕ Если условное выражение возвращает **false** при первом проходе цикла, код в фигурных скобках выполняться не будет.

Подобно операторам if, циклы **for** также могут быть вложенными. Следующий листинг содержит код прохода по каждой координате сетки 10x10. Для каждой итерации первого цикла имеют место 10 итераций вложенного цикла. В результате вложенный цикл будет выполняться 100 раз.

```
<html>
<head>
<title>Вложенные
</head>
<body>
<script type="text/javascript">
<!--
  document.write("Все координаты x,y (0,0) и (9,9) :<br>");
  for (var x = 0; x < 10; ++x) {
    for (var y = 0; y < 10; ++y) {
      document.write("("+x+","+У+"),");
    }
    document.write('<br>');
  }
  1
  document.write("<br>После завершения цикла x равно :" + x);
  document.write("<br>После завершения цикла y равно :" + y);
//-->
</script>
</body>
</html>
```

Оператор for..in

Конструкция **for..in** предоставляет возможность выполнения набора операторов для каждого свойства объекта. Цикл **for..in** можно использовать с любым объектом JavaScript, независимо от его свойств. Если объект не имеет свойств, цикл не выполняется. Цикл **for..in** работает также и с пользовательскими объектами. Переменная пользовательского объекта JavaScript рассматривается как свойство, и поэтому для каждой из них выполняется цикл.

Синтаксис выглядит так:

for (свойство in объект)

операторы г

Тут *свойство* — это строковый литерал, сгенерированный JavaScript. При каждом проходе цикла *свойству* присваивается имя следующего свойства, содержащееся в *объекте,* и так до тех пор, пока не переберутся все свойства.

В следующем листинге такая функциональность применяется для отображения имен и значений всех свойств объекта **Document**.

```
<html>
<head>
<title>JavaScript Unleashed</title>
</head>
<body>
<script language="JavaScript1.1" type="text/javascript">
<!--
  // Объявление переменных
  var anObject = document;
  var propertyInfo = "";
  for (var propertyName in anObject) {
    propertyInfo = propertyName + " — " + anObject[propertyName];
    document.write(propertyInfo + "<br>");
  }
</script>
</body>
</html>
```

ПРЕДУПРЕЖДЕНИЕ

Цикл **for..in** работает только в версии JavaScript 1.1 и выше.

Оператор while

Оператор **while** действует подобно циклу **for,** однако не включает в себя функции инициализации приращения **переменных** во время их объявления.

Переменные необходимо объявлять **заранее,** а инкремент или **декремент** их определять в рамках блока операторов. Рассмотрим синтаксис **while:**

while (условие) {

операторы ; }

```
<html>
<head>
```

```
<title>JavaScript Unleashed</title>
</head>
<body>
<script type="text/javascript">
<!--
// Объявление переменных
   var i = 0;
   var result = 0;
   var status = true;
   document.write("0");
   while(status){
      result += ++i;
      document.write(" + " + i);
      if (i = 10 )) {
         status = false;
      }
   }
   document.writeln(" = " + result);
//-->
</script>
</body>
</html>
```

Оператор do...while

Начиная с JavaScript 1.2, предлагается конструкция **do..while.** Она работает подобно оператору **while** за исключением того, что условное выражение не проверяется вплоть до завершения первой итерации. Этот способ гарантирует, что набор операторов, находящихся в пределах фигурных скобок, будет выполнен по крайней мере один раз.

```
<html>
<head>
<title>JavaScript Unleashed</title>
<head>
<body>
<script language="JavaScript1.2" type="text/javascript">
<!--
   //Declare variables
   var userEntry = 0;
```

```
  var x = 1;
  do {
    document.writeln(x);
    x++;
  } while(x <= userEntry);
//-->
</script>
</body >
</html>
```

Используя циклы, стоит обратить внимание на то, что по умолчанию цикл не прекращается, а будет выполняться до тех пор, пока определенное условие не будет равно **false.** Однако, иногда может потребоваться выйти из цикла до того, как он дойдет до закрывающей фигурной **скобки.** Подобное достигается за счет помещения в блок операторов внутри цикла **break** или **continue.**

Оператор **break** завершает выполнение цикла, в то время как **continue** пропускает оставшиеся операторы текущей итерации, вычисляет очередное значение выражения цикла (если таковое существует) и начинает выполнение следующей итерации. Различия между этими двумя операторами продемонстрированы в следующем листинге, который содержит достаточно тривиальный сценарий вычисления приближенного целочисленного значения квадратного корня числа n.

```
<html>
<head>
<title>Операторы break и cantinue</title>
</head>
<body>
<script type="text/javascript">
<!--
  // Объявление переменных
  var highestNum = 0;
  var n = 175;    // Тестовое значение
  for (var i = 0; i < n; ++i) {
    document.write(i + "<br>");
    if(n < 0) {
      document.write ("n не может быть отрицательным.");
      break;
```

```
        }
        if (i * i <= n) {
            highestNum = i;
            continue ;
        }
        document.write("<br>Сделано!") ;
        break;
    }
    document, write ("<br>Целочисленное значение, не превышающее
квадратный корень");
    document.write(" из " + n + " = " highestNum);
//-->
</script>
</body>
</html>
```

4.3. МЕТКИ

Начиная с версии JavaScript 1.2, в языке появился метод обозначения более специфического места перехода при использовании операторов **break и continue** — метки. Метку можно помещать перед любой управляющей структурой, которая содержит другие операторы. Это позволяет перейти из рамок условного оператора либо цикла на совершенно определенное место программы. Пример применения меток показан в следующем листинге.

```
<html><head><title>Использование меток</title></head><body>
<script language=" JavaScript1.2" type="text/javascript">
<!--
    // Объявление переменных
    var stopX = 3;
    var stopY = 8 ;
    document.write ("Все пары x.y между (0,0) и (")
    document.write (stopX + "," + stopY + ") :<br>")
loopX:
    for (var x = 0 ; x < 10; ++x) {
        for (var y = 0; y < 10; ++y) {
            document.write (" ("+x + "," + y +") ") ;
            if((x = stopX) is (y = stopY) ){
                break loopX;
```

```
            }
        }
        document.write("<br>");
    }
    document.write ("<br>После завершения цикла x равно: " + x);
    document.writeln <"<br>После завершения цикла y равно : " + y) ;
//-->
</script></body></html>
```

4.4. ОПЕРАТОР WITH

Оператор **with** применяется во избежание многократного повторения ссылки на объект при доступе к его свойствам и методам. Любые свойства или методы в блоке **with,** которые JavaScript не может опознать, ассоциируются с объектом, указанным в этом блоке. Синтаксис **with** таков:

```
with (объект) {
  операторы
}
```

```
<html>
<head>
<title>Использование with</title>
</head>
<body>
<script type="text/javascript">
<!--
    with(document) {
        write("Привет!");
        write("<br>Заголовок этого документа выглядит как
\""+title+"\".");
        write("<br>URL этого документа выглядит как \"" + URL + "\".">
;
        write("Теперь от префикса можно избавиться раз и навсегда!");
    }
//-->
</script>
</body>
</html>
```


Библиотека ГГТУ им.П.О.Сухого

4.5. ОПЕРАТОР SWITCH

Оператор switch используется для сравнения одного значения с большим количеством других. На первый взгляд упомянутая задача решается только за счет применения большого числа условных операторов, однако оператор switch тут будет гораздо уместнее. Он быстрее читается и позволяет определить заданный блок инструкций, выполняющийся в случае, когда соответствие не найдено.

```
<html><head><title>JavaScript Unleashed</title></head><body>
<script language="JavaScript1.2" type="text/javascript">
<!--
  // Объявление переменных
  var request = "Name";
  switch(request) {
    case "Logo":
      document.write('<img src="logo.gif" alt="Logo">');
      document.write("<br>");
      break;
    case "Name":
      document.write("Software Inc.");
      document.write("<br>");
      break;
    case "Products":
      document.write("MyEditor");
      document.write("<br>");
      break;
    default:
      document.write("www.mysite.com");
      break;
  }
//-->
</script></body></html>
```

РАЗДЕЛ 5. ФУНКЦИИ, МАССИВЫ, ОБЪЕКТЫ

Язык программирования не может обойтись без механизма многократного использования кода программы. Такой механизм обеспечивается *процедурами* или *функциями*. В JavaScript *функция*

Библиотека ГГТУ им.П.О.Сухого

выступает в качестве одного из основных *типов данных*. Одновременно с этим в JavaScript определен класс объектов Function.

В общем случае любой *объект* JavaScript определяется через функцию. Для создания *объекта* используется конструктор, который в свою очередь вводится через Function. Таким образом, с *функциями* в JavaScript связаны следующие ключевые вопросы:

- функция как тип данных;
- функция как объект;
- функция как конструктор объектов.

5.1. ФУНКЦИЯ КАК ТИП ДАННЫХ

Определяют *функцию* при помощи ключевого слова function:

```
function f(arg1,arg2,...)
{
    /* тело функции */
}
```

Здесь следует обратить внимание на следующие моменты. Во-первых, function определяет *переменную* с именем f. Эта переменная имеет тип function:

```
document.write('Тип переменной f: '+ typeof(f));
// Будет выведено: Тип переменной f: function
```

Если требуется, чтобы функция возвращала некоторое значение, то в ее теле используется оператор возврата **return** с указанием справа от него того, что следует возвратить. В качестве возвращаемой величины может выступать любое выражение: простое значение, имя переменной или вычисляемое выражение. Оператор return может встречаться в коде функции несколько раз. Впрочем, возвращаемую величину, а также сам оператор return можно и не указывать. В этом случае функция ничего не будет возвращать.

```
function f(arg1,arg2,...)
{
    /* операторы */
    return /* значение */;
}
```


Библиотека ГГТУ им.П.О.Сухого

5.2. РЕКУРСИВНЫЕ ФУНКЦИИ

Функции JavaScript могут быть рекурсивными, т.е. вызывать сами себя. Вычисление факториала — лучший способ демонстрации рекурсивного процесса. Ввиду того что JavaScript не имеет встроенной операции факториала, их придется вычислять самостоятельно, причем с использованием механизма рекурсивных функций. Воспользуемся рекуррентной формулой **n!=(n-1)!*n и 1!=1**

```
function getFactorial(n) {
    var result;
    if (n > 0) {
            result = n * getFactorial (n)
    }else if (n=0) {
            result = 1;
    } else {
            result = null;
    }
    return (result)
}
```

5.3. ФУНКЦИЯ КАК ПЕРЕМЕННАЯ

Во-вторых, эта переменная, как и любая другая, имеет значение — свой исходный текст:

```
var i=5;
function f(a,b,c)
{
 if (a>b) return c;
}

document.write('Значение переменной i: '+ i.valueOf());
// Будет выведено:
// Значение переменной i: 5

document.write('Значение переменной f:<BR>'+ f.valueOf());
// Будет выведено:
// Значение переменной f:
// function f(a,b,c)
```


Библиотека ГГТУ им.П.О.Сухого

```
// {
//  if (a>b) return c;
// }
```

Как видим, метод valueOf() применим как к числовой переменной i, так и к переменной f, и возвращает их значение.

Более того, значение переменной f можно присвоить другой переменной, тем самым создав "синоним" функции f:

```
function f(a,b,c)
{
 if (a>b) return c;
 else return c+8;
}

var g = f;
alert('Значение f(2,3,2): '+ f(2,3,2) );
alert('Значение g(2,3,2): '+ g(2,3,2) );

// Будет выведено:
// Значение f(2,3,2): 10
// Значение g(2,3,2): 10
```

Этим приемом удобно пользоваться для сокращения длины кода. Например, если нужно много раз вызвать метод document.write(), то можно ввести переменную: **var W = document.write**

(обратите внимание — без скобок!), а затем вызывать: W('<H1>Лекция</H1>').

Коль скоро функцию можно присвоить переменной, то ее можно передать и в качестве аргумента другой функции.

```
function kvadrat(a)
{   return a*a;   }

function polinom(a,k)
{ return k(a)+a+5;}

alert(polinom(3,kvadrat));
// Будет выведено: 17
```

Все это усиливается при использовании функции **eval**(), которая в качестве аргумента принимает строку, которую рассматривает как

Библиотека ГГТУ им.П.О.Сухого

последовательность операторов JavaScript (блок) и выполняет этот блок. В качестве иллюстрации приведем скрипт, который позволяет вычислять функцию f(f(...f(N)...)), где число вложений функции f() задается пользователем.

```
<html>
<head>
<title>JavaScript Unleashed</title>
</head>
<body>
<script>
   function kvadrat(a){
      return a*a;
   }

   function SuperPower(){
      var N = parseInt(document.f.n.value),
      K = parseInt(document.f.k.value), L = R = '';
      for(i=0; i<K; i++){
         L+='kvadrat(';
         R+=')';
      }
   return eval(L+N+R);
   }
</script>
<form name="f">
 введите аргумент (число):
 <input name="n"><br>
 сколько раз возвести его в квадрат?
 <input name="k"><br>
 <input type=button value="возвести" onclick="alert(superpower());">
</form>
</body>
</html>
```

Пример. Многократное вложение функции kvadrat() в себя

Библиотека ГГТУ им. П.О. Сухого

5.4. ФУНКЦИЯ КАК ОБЪЕКТ

У любого *типа данных* JavaScript существует *объектовая* "обертка" (wrapper), которая позволяет применять методы типов данных к переменным и литералам, а также получать значения их свойств. Например, длина строки символов определяется свойством length. Аналогичная "обертка" есть и у функций — это класс объектов Function.

Например, увидеть значение функции можно не только при помощи метода valueOf(), но и используя метод toString():

```
function f(x,y)
{
 return x-y;
}
document.write(f.toString());
```

Результат распечатки:

```
function f(x,y) { return x-y; }
```

Свойства же функции как объекта доступны программисту только тогда, когда они вызываются внутри этой функции.

Наиболее часто используемыми свойствами являются:

arguments[] - массив (коллекция) аргументов функции (arguments[]),

length - его длина (length),

caller - имя функции, вызвавшей данную функцию (caller),

prototype - прототип *(prototype)*.

Рассмотрим пример использования списка аргументов функции и его длины:

```
function my_sort()
{
 a = new Array(my_sort.arguments.length);
 for(i=0;i<my_sort.arguments.length;i++)
   a[i] = my_sort.arguments[i];
 return a.sort();
}

b = my_sort(9,5,7,3,2);
document.write(b);
// Будет выдано: 2,3,5,7,9
```


Библиотека ГГТУ им. П.О. Сухого

Чтобы узнать, какая функция вызвала данную функцию, используется свойство caller. Возвращаемое ею значение имеет тип function. Пример:

```
function s()
{ document.write(s.caller+"<BR>"); }
function M()
{ s(); return 5; }
function N()
{ s(); return 7; }
M();   N();
```

Результат исполнения:

```
function M() { s(); return 5; }
function N() { s(); return 7; }
```

5.5. МАССИВЫ. ФУНКЦИИ ДЛЯ РАБОТЫ С МАССИВАМИ

Массивы делятся на

- встроенные (document.links[], document.images[] и т.п. — их еще называют коллекциями) и
- определяемые пользователем (автором документа). Коллекции будут обсуждаться в следующей лекции.

Для массивов определено несколько методов: join(), reverse(), sort() и другие, а также свойство length, которое позволяет получить число элементов массива.

Для определения массива пользователя существует специальный конструктор Array. Если ему передается единственный аргумент, причем целое неотрицательное число, то создается незаполненный массив соответствующей длины. Если же передается один аргумент, не являющийся числом, либо более одного аргумента, то создается массив, заполненный этими элементами:

```
a = new Array();
// пустой массив (длины 0)

b = new Array(10);
// массив длины 10

c = new Array(10,'Привет');
```


Библиотека ГГТУ им.П.О.Сухого

```
// массив из двух элементов: числа и строки

d = [5, 'Тест', 2.71828, 'Число е'];
// краткий способ создать массив из 4 элементов
```

Элементы массива нумеруются **с нуля**. Поэтому в последнем примере значение d[0] равно 5, а значение d[1] равно 'Тест'. Как видим, массив может состоять из разнородных элементов. Массивы не могут быть многомерными, однако ничто не мешает завести массив, элементами которого будут тоже массивы.

Методы join() и split()

Метод join() позволяет объединить элементы массива в одну строку. Он является обратным к методу split(), который разрезает объект типа String на куски и составляет из них массив. Кстати, метод split() демонстрирует тот факт, что массив можно получить и без конструктора массива.

Синтаксис

```
// split – это метод строки
a = s.split ([разделитель]);
// join – это метод массива
s = a.join ([разделитель]);
```

Рассмотрим пример преобразования локального URL в глобальный URL, где в качестве адреса сервера будет выступать www.gstu.by.

```
<html><head>
<title>JavaScript Unleashed</title>
</head><body>
<script>
   // Пусть в переменной localURL хранится локальный URL
некоторого файла:
   localURL = "file:///C:/department/internet/js/2/2.html"
   // Разрежем строку в местах вхождения комбинации символов ":/",
   // выполнив команду:
   b = localURL.split(':/').
   // Получим массив:
   b[0] = "file";
   b[1] = "//C";
   b[2] = "department/internet/js/2/2.html";
```


Библиотека ГГТУ им. П.О. Сухого

```
   // Заменяем 0-й и 1-й элементы на требуемые:
   b[0] = "http:";
   b[1] = "/www.gstu.by";
   // Наконец, склеиваем полученный массив, вставляя косую черту в
местах склейки:
   globalURL = b.join("/").
   // В итоге мы получаем требуемый глобальный URL — значение
globalURL
   globalURL = "http://www.gstu.by/department/internet/js/2/2.html"
</script></body></html>
```

Метод reverse()

Метод reverse() применяется для изменения порядка элементов массива на противоположный. Предположим, массив упорядочен следующим образом:

a = new Array('мать','видит','дочь');

Упорядочим его в обратном порядке, вызвав метод **a.reverse().** Тогда новый массив a будет содержать:

a[0]='дочь';
a[1]='видит';
a[2]='мать';

Метод sort()

Метод **sort()** интерпретирует элементы массива как строковые литералы и сортирует массив в алфавитном (т.н. лексикографическом) порядке. Обратите внимание: метод sort() меняет массив. В предыдущем примере, применив **a.sort(),** мы получим на выходе:

a[0]='видит';
a[1]='дочь';
a[2]='мать';

Однако, это неудобно, если требуется отсортировать числа, поскольку согласно алфавитному порядку 40 идет раньше чем 5. Для этих целей у метода sort() имеется необязательный аргумент, являющийся именем функции, согласно которой требуется отсортировать массив, т.е. в этом случае вызов метода имеет вид:

Библиотека ГГТУ им. П.О. Сухого

a.sort(myfunction). Эта функция должна удовлетворять определенным требованиям:

- у нее должно быть ровно два аргумента;
- функция должна возвращать число;
- если первый аргумент функции должен считаться меньшим (большим, равным) чем второй аргумент, то функция должна возвратить отрицательное (положительное, ноль) значение.

Например, если нам требуется сортировать числа, то мы можем описать следующую функцию:

```
function compar(a,b)
{
 if(a < b)  return -1;
 if(a > b)  return  1;
 if(a == b) return  0;
}
```

Теперь, если у нас есть массив

b = new Array(10,6,300,25,18);,

то можно сравнить результаты сортировки без аргумента и с функцией compar в качестве аргумента:

```
document.write("Алфавитный порядок:<BR>");
document.write(b.sort());
document.write("<BR>Числовой порядок:<BR>");
document.write(b.sort(compar));
```

В результате выполнения этого кода получим следующее:

Алфавитный порядок:
10,18,25,300,6
Числовой порядок:
6,10,18,25,300

Обратите внимание: метод sort() интерпретирует элементы массива как строки (и производит лексикографическую сортировку), но не преобразует их в строки. Если в массиве были числа, то они числами и останутся. В этом легко убедиться, если в конце последнего примера выполнить команду document.write(b[3]+1): результат будет 26 (т.е. 25+1), а не 251 (т.е. "25"+1).

Библиотека ГГТУ им.П.О.Сухого

5.6. ОБЪЕКТЫ

Объект — это главный тип данных JavaScript. Любой другой тип данных имеет объектовую "обертку" (wrapper). Это означает, что прежде чем можно будет получить доступ к значению переменной того или иного типа, происходит конвертирование переменной в объект, и только после этого выполняются действия над значением. Тип данных Object сам определяет объекты.

В сценарии JavaScript могут использоваться объекты нескольких видов:

- **клиентские объекты**, входящие в модель DOM, т.е. отвечающие тому, что содержится или происходит на Web-странице в окне браузера. Они создаются браузером при разборе (парсинге) HTML-страницы. Примеры: window, document, location, navigator и т.п.
- **серверные объекты**, отвечающие за взаимодействие клиент-сервер. Примеры: Server, Project, Client, File и т.п. Серверные объекты в этом курсе рассматриваться не будут.
- **встроенные объекты**. Они представляют собой различные типы данных, свойства, методы, присущие самому языку JavaScript, независимо от содержимого HTML-страницы. Примеры: встроенные классы объектов Array, String, Date, Number, Function, Boolean, а также встроенный объект Math.
- **пользовательские объекты**. Они создаются программистом в процессе написания сценария с использованием конструкторов типа объектов (класса). Например, можно создать свои классы Cat и Dog. Создание и использование таких объектов будет рассмотрено далее в этой лекции.

Операторы работы с объектами for ... in ...

Оператор for(переменная in объект) позволяет "пробежаться" по свойствам объекта. Рассмотрим пример (об объекте document см. ниже):

```
for(v in document)
document.write("document."+v+" = <B>"+
document[v]+"</B><BR>");
```

Результатом работы этого скрипта будет длинный список свойств объекта document, мы приведем лишь его начало (полностью получите его самостоятельно):

```
alinkColor = #0000ff
bgColor = #ffffff
mimeType = HTML Document
defaultCharset = windows-1251
lastModified = 07/16/2002 21:22:53
onclick = null
links = [object]
...
```

Примечание Попробуйте запустить этот скрипт в разных браузерах — и Вы увидите, что набор свойств у объекта document различный в различных браузерах. Аналогичная ситуация со многими объектами модели DOM, о которой пойдет речь ниже. Именно поэтому приходится постоянно заботиться о так называемой кроссбраузерной совместимости при программировании динамических HTML-документов.

with

Оператор with задает объект по умолчанию для блока операторов, определенных в его теле. Синтаксис его таков:

with (объект) оператор;

Все встречающиеся в теле этого оператора свойства и методы должны быть либо записанными полностью, либо они будут считаться свойствами и методами объекта, указанного в операторе with. Например, если в документе есть форма с именем anketa, а в ней есть поля ввода с именами age и speciality, то мы можем воспользоваться оператором with для сокращения записи:

```
with (document.anketa)
{
 age.value=35;
 speciality.value='программист';
 window.alert(length);
 submit();
}
```

Оператором with полезно пользоваться при работе с объектом Math, используемым для доступа к математическим функциям и

Библиотека ГГТУ им.П.О.Сухого

константам. Например, внутри тела оператора with(Math) можно смело писать: sin(f)*cos(h+PI/2); без оператора with пришлось бы указывать Math три раза: Math.sin(f)*Math.cos(h+Math.PI/2)

Клиентские объекты

Для создания механизма управления страницами на клиентской стороне используется объектная модель документа (DOM — Document Object Model). Суть модели в том, что каждому HTML-контейнеру соответствует объект, который характеризуется тройкой:

свойства

методы

события

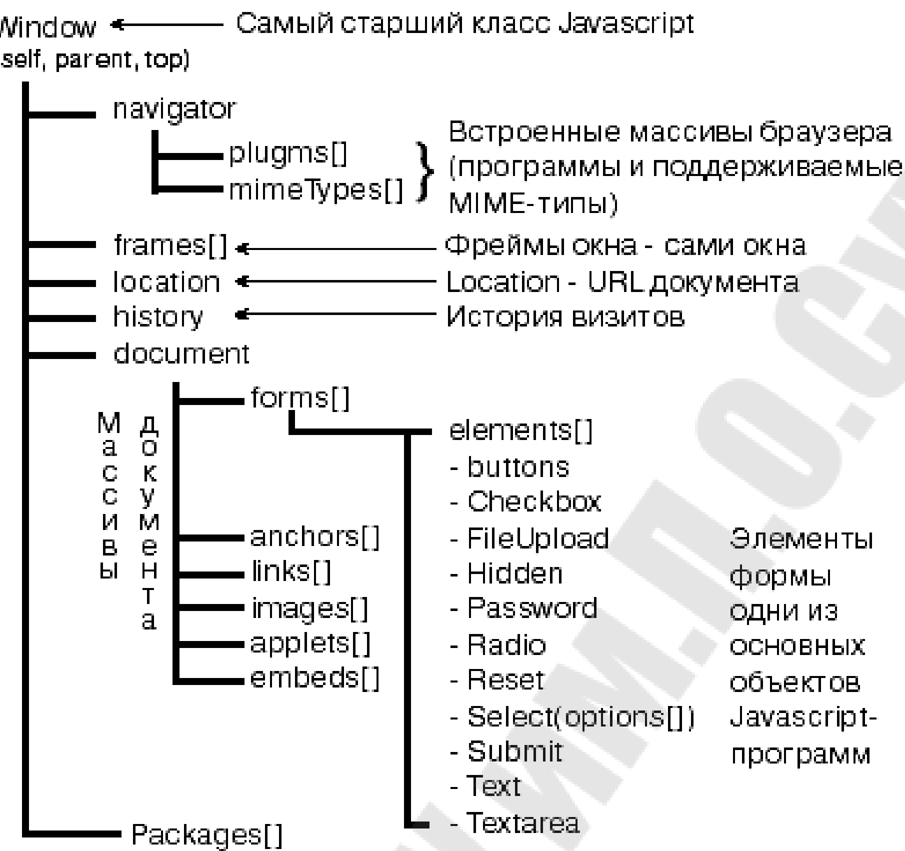
Объектную модель можно представить как способ связи между страницами и браузером. Объектная модель документа — это представление объектов, их методов, свойств и событий, которые присутствуют и происходят в программном обеспечении браузера, в виде, удобном для работы с ними из кода HTML и исходного текста сценария на странице. Мы можем с ее помощью сообщать наши пожелания браузеру и далее — посетителю страницы. Браузер выполнит наши команды и соответственно изменит страницу на экране.

Объекты с одинаковым набором свойств, методов и событий объединяются в классы однотипных объектов. Классы — это описания возможных объектов. Сами объекты появляются только после загрузки документа браузером или как результат работы программы. Об этом нужно всегда помнить, чтобы не обратиться к объекту, которого нет.

Иерархия классов DOM

Объектно-ориентированный язык программирования предполагает наличие иерархии классов объектов. В JavaScript такая иерархия начинается с класса объектов window

Библиотека ГГТУ им. П.О.Сухого



Вообще говоря, JavaScript не является классическим объектным языком (его еще называют облегченным объектным языком). В нем нет наследования и полиморфизма. Имеется лишь отношение "объект А содержит объект В" (которое и проиллюстрировано на рис. 3.1). Оно не является иерархией классов в буквальном смысле. Действительно, нахождение класса window в этой иерархии выше класса history не означает, что всякий объект типа history является объектом типа window и наследует все его свойства и методы, как это понималось бы в стандартных объектно-ориентированных языках. В JavaScript же это отношение означает лишь то, что объект history является свойством объекта window, а значит, чтобы получить к нему доступ, нужно воспользоваться "точечной нотацией": window.history.

У объектов DOM некоторые свойства обязательно присутствуют, тогда как наличие других зависит от Web-страницы. Например, объект window всегда имеет в качестве своих свойств объекты location и history, т.е. это обязательные свойства. Если HTML-страница содержит контейнер <BODY>, то у объекта window будет присутствовать в качестве свойства объект document. Если HTML-страница содержит контейнер <FRAMESET> со вложенными в него контейнерами <FRAME>, то у объекта window будут присутствовать

Библиотека ГГТУ им.П.О.Сухого

в качестве свойств имена фреймов, например window.f1. Последние, как мы увидим в будущих лекциях, сами являются объектами класса window, и для них в свою очередь справедливо все вышесказанное.

5.7. КОЛЛЕКЦИИ

Коллекция — это структура данных JavaScript, похожая на массив. Отличие коллекции от массивов заключается в том, что массивы программист создает сам в коде программы и заполняет их данными; коллекции же создаются браузером и "населяются" объектами, связанными с элементами Web-страницы. Коллекцию можно рассматривать как другой, зачастую более удобный способ доступа к объектам Web-страницы.

Нумеруются элементы коллекции, начиная с нуля, в порядке их появления в исходном HTML-файле. Доступ к элементам коллекций осуществляется либо по индексу (в круглых или квадратных скобках), либо по имени (тоже в круглых или квадратных скобках, либо через точку), например:

```
window.document.forms[4]     // 5-я форма на странице
window.document.forms(4)     // равносильно предыдущему

window.document.forms['mf']  // форма с именем 'mf'
window.document.forms('mf')  // равносильно предыдущему

window.document.forms.mf     // равносильно предыдущему
window.document.mf           // равносильно предыдущему
```

Способы в 3-4 строчках удобны, когда имя элемента коллекции хранится в качестве значения переменной. Например, если мы задали var w="mf", то мы можем обратиться к форме с именем "mf" как window.document.forms[w]. Именно так мы поступили выше в разделе про оператор for...in, когда выписывали список всех свойств объекта document.

Как и у обычных массивов, у коллекций есть свойство length, которое позволяет узнать количество элементов в коллекции. Например, document.images.length.

Библиотека ГГТУ им. П.О.Сухого

Перечислим основные коллекции в объектной модели документа.

window.frames[]	Все фреймы — т.е. объекты, отвечающие контейнерам <FRAME>
document.all[]	Все объекты, отвечающие контейнерам внутри контейнера <BODY>
document.anchors[]	Все якоря — т.е. объекты, отвечающие контейнерам <A>
document.applets[]	Все апплеты — т.е. объекты, отвечающие контейнерам <APPLET>
document.embeds[]	Все вложения — т.е. объекты, отвечающие контейнерам <EMBED>
document.forms[]	Все формы — т.е. объекты, отвечающие контейнерам <FORM>
document.images[]	Все картинки — т.е. объекты, отвечающие контейнерам <IMG>
document.links[]	Все ссылки — т.е. объекты, отвечающие контейнерам <A HREF="..."> и <AREA HREF="...">
document.f.elements[]	Все элементы формы с именем f — т.е. объекты, отвечающие контейнерам <INPUT> и <SELECT>
document.f.s.options[]	Все опции (контейнеры <OPTION>) в контейнере <SELECT NAME=s> в форме <FORM NAME=f>
navigator.mimeTypes[]	Все типы MIME, поддерживаемые браузером (список см. на сайте IANA)
function_name.arguments[]	Все аргументы, переданные функции function_name() при вызове

Свойства

Многие HTML-контейнеры имеют атрибуты. Как мы уже знаем, каждому контейнеру соответствует объект. При этом соответствии атрибутам отвечают свойства объекта. Соответствие между атрибутами HTML-контейнеров и свойствами DOM-объектов не всегда прямое. Обычно каждому атрибуту отвечает некоторое свойство объекта. Но, во-первых, название этого свойства не всегда легко угадать по названию атрибута, а во-вторых, у объекта могут быть свойства, не имеющие аналогов среди атрибутов. Кроме того,

Библиотека ГГТУ им. П.О.Сухого

как мы знаем, атрибуты являются регистро-независимыми, как и весь язык HTML, тогда как свойства объектов нужно писать в точно определенном регистре символов.

Например, контейнер якоря <A ...>...</A> имеет атрибут HREF, который превращает его в гипертекстовую ссылку:

```
<A HREF="http://intuit.ru/">intuit</A>
```

Данной гиперссылке соответствует объект (класса URL) — document.links[0], если предполагать, что это первая ссылка в нашем документе. Тогда атрибуту HREF будет соответствовать свойство href этого объекта. К свойству объекта можно обращаться с помощью точечной нотации: объект.свойство. Например, чтобы изменить адрес, на который указывает эта ссылка, мы можем написать:

```
document.links[0].href='http://ya.ru/';
```

Методы

В терминологии JavaScript методы объекта определяют функции, с помощью которых выполняются действия с этим объектом, например, изменение его свойств, отображения их на web-странице, отправка данных на сервер, перезагрузка страницы и т.п.

Например, если у нас есть ссылка <A HREF="http://gstu.by/">intuit</A> (будем считать, она первая в нашем документе), то у соответствующего ей объекта document.links[0] есть метод click(). Его вызов в любом месте JavaScript-программы равносилен тому, как если бы пользователь кликнул по ссылке, что демонстрирует пример:

```
<A HREF=" http://gstu.by/">GSTU</A>
<SCRIPT> document.links[0].click(); </SCRIPT>
```

При открытии такой страницы пользователь сразу будет перенаправлен на сайт Технического университета. Обратите внимание, что скрипт написан после ссылки. Если бы мы написали его до ссылки, то поскольку в этот момент ссылки (а значит и объекта) еще не существует, браузер выдал бы сообщение об ошибке.

Некоторые методы могут применяться неявно. Для всех объектов определен метод преобразования в строку символов: toString(). Например, при сложении числа и строки число будет преобразовано в строку:

```
"25"+5 = "25"+(5).toString() = "25"+"5" = "255"
```

Аналогично, если обратиться к объекту window.location (рассматриваемом в следующей лекции) в строковом контексте,

Библиотека ГГТУ им. П.О. Сухого

скажем, внутри вызова document.write(), то неявно будет выполнено это преобразование, и программист этого не заметит, как если бы он распечатывал не объект, а строку:

```
<SCRIPT>
   document.write('Неявное преобразование: ');
   document.write(window.location);
   document.write('<BR>Явное преобразование: ');
   document.write(window.location.toString());
</SCRIPT>
```

Тот же эффект можно наблюдать для встроенных объектов типа Date:

```
<SCRIPT>
   var d = new Date();
   document.write('Неявное преобразование: ');
   document.write(d);
   document.write('<BR>Явное преобразование: ');
   document.write(d.toString());
</SCRIPT>
```

События

Кроме методов и свойств, объекты характеризуются событиями. Собственно, суть программирования на JavaScript заключается в написании обработчиков этих событий. Например, с объектом типа button (контейнер INPUT типа button — "кнопка") может происходить событие Click, т.е. пользователь может нажать на кнопку. Для этого атрибуты контейнера INPUT расширены атрибутом обработки этого события — onClick. В качестве значения этого атрибута указывается программа обработки события, которую должен написать на JavaScript автор HTML-документа:

```
<INPUT TYPE=button VALUE="Нажать"
 onClick="alert('Пожалуйста, нажмите еще раз')">
```

Обработчики событий указываются в специально созданных для этого атрибутах у тех контейнеров, с которыми эти события связаны. Например, контейнер BODY определяет свойства всего документа, поэтому обработчик события "завершена загрузка всего документа" указывается в этом контейнере как значение атрибута onLoad.

Примеры событий: нажатие пользователем кнопки в форме, установка фокуса в поле формы или увод фокуса из нее, изменение введенного в поле значения, нажатие кнопки мыши, отпускание

Библиотека ГГТУ им.П.О.Сухого

кнопки мыши, щелчок кнопкой мыши на объекте (ссылке, поле, кнопке, изображении и т.п.), двойной щелчок кнопкой мыши на объекте, перемещение указателя мыши, выделение текста в поле ввода или на странице и другие.

5.8. ПОЛЬЗОВАТЕЛЬСКИЕ ОБЪЕКТЫ

Мы не будем очень подробно вникать во все эти моменты, так как при программировании на стороне браузера чаще всего обходятся встроенными средствами JavaScript. Но поскольку все эти средства — объекты, нам нужно понимать, с чем мы имеем дело.

Понятие пользовательского объекта

Сначала рассмотрим пример определенного пользователем объекта класса Rectangle, потом выясним, что же это такое:

```
function Rectangle(a,b,c,d)
{
this.x0 = a;
this.y0 = b;
this.x1 = c;
this.y1 = d;

this.area = new Function(
 "return Math.abs((this.x1-this.x0)*(this.y1-this.y0))");
}
r = new Rectangle(0,0,30,50);
```

Создается четыре переменных: x0, y0, x1, y1 — это свойства объекта r. К ним можно получить доступ только в контексте объекта данного класса, например:

```
up_left_x = r.x0;
up_left_y = r.y0;
```

Кроме свойств, внутри конструктора Rectangle мы определили объект area класса Function(), применив встроенный конструктор языка JavaScript. Это методы объекта класса Rectangle. Вызвать эту функцию можно тоже только в контексте объекта класса Rectangle:

```
sq = r.area();
```


Библиотека ГГТУ им. П.О.Сухого

Таким образом, объект — это совокупность свойств и методов, доступ к которым можно получить, только создав при помощи конструктора объект данного класса и использовав его контекст.

На практике довольно редко приходится иметь дело с объектами, созданными программистом. Дело в том, что объект создается функцией-конструктором, которая определяется на конкретной странице и, следовательно, все, что создается в рамках данной страницы, не может быть унаследовано другими страницами. Нужны очень веские основания, чтобы автор Web-узла занялся разработкой библиотеки пользовательских классов объектов. Гораздо проще писать функции для каждой страницы.

Прототип

Обычно мы имеем дело со встроенными объектами JavaScript, такими как Data, Array и String. Собственно, почти все, что изложено в других разделах курса (кроме иерархии объектов DOM) — это обращение к свойствам и методам встроенных объектов. В этом смысле интересно одно свойство объектов, которое носит название prototype. Прототип — это другое название конструктора объекта конкретного класса. Например, если мы хотим добавить метод к объекту класса String, то мы можем это сделать следующим образом:

```
String.prototype.out = new Function("a", "a.write(this)");
var s = "Привет!";
s.out(document);
// Будет выведено: Привет!
```

Для объявления нового метода для объектов класса String мы применили конструктор Function. Есть один существенный нюанс: новыми методами и свойствами будут обладать только те объекты, которые порождаются после изменения прототипа объекта. Все встроенные объекты создаются до того, как JavaScript-программа получит управление, что существенно ограничивает применение свойства prototype.

Методы объекта Object

Object — это класс, элементами которого являются любые объекты JavaScript. У всех объектов этого класса есть общие методы. Таких методов мы рассмотрим три: toString(), valueOf() и assign().

Библиотека ГГТУ им. П.О.Сухого

Метод **toString()** осуществляет преобразование объекта в строку символов (строковый литерал). Он используется в JavaScript-программах повсеместно, но в основном неявно. Например, при выводе числа или строковых объектов. Интересно применение toString() к функциям:

Аналогично ведет себя и метод **valueOf()**, позволяющий получить значение объекта. В большинстве случаев он работает подобно методу toString(), особенно если нужно выводить значение на страницу.

В отличие от двух предыдущих методов, метод **assign()** позволяет не прочитать, а переназначить какое-либо свойство и метод объекта. Следует заметить, что этот метод работает не во всех браузерах и не со всеми объектами. В общем случае **оператор объект.свойство = значение** равносильно оператору объект.свойство.assign(значение). Например, следующие операторы равносильны — они перенаправляют пользователя на новую страницу:

```
window.location = "http://gstu.by/";
window.location.assign("http://gstu.by /");
```

РАЗДЕЛ 6. СТРОКИ И РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

6.1. ОБЪЕКТ STRING

Строки — это фундаментальное понятие любого языка программирования. Строка, представляющая собой набор алфавитно-цифровых символов, может быть либо текстовой (литеральной) строкой, такой как "Не морочьте голову", либо строковой переменной, например, thePhrase.

Строки относятся к одному из важнейших типов данных, поэтому просто необходимо уметь выделять данные из строки получать о них информацию. Например, для определения размера строки можно воспользоваться свойством length. В рассмотренном примере возвращается 13:

```
var myString = new String ("Это мой день.");
var len = myString.length;
```

Методы класса String

charAt(i)	Возвращает символ строки, расположенный по заданному индексу.

Библиотека ГГТУ им. П.О.Сухого

CharCodeAt(i)	Возвращает код символа ISO-Latin-1, расположенного по индексу, переданному в метод.
concat(s1,s2)	Связывает две переданные строки в одну новую.
fromCharCode(n)	Возвращает строковое значение числа ISO-Latin-1, переданное методу.
indexOf(s,n)	Возвращает позицию первого вхождения строки, переданной методу, в экземпляре объекта String.
lastIndexOf(s,n)	Возвращает позицию последнего вхождения строки, переданной методу, в экземпляре объекта String.
match()	Возвращает массив, основанный на регулярном выражении, переданном в метод.
replace(regExp)	Выполняет операцию поиска и замены, заданную регулярным выражением, и заменяет на строку, переданную в метод.
search(s)	Возвращает позицию совпадения с заданной строкой в экземпляре объекта String. Если строка не найдена, возвращается значение -1.
slice(m,n)	Возвращает часть строки между заданными начальной и конечной позициями. При передаче отрицательного числа отсчет будет выполняться с конца строки.
split(s)	Выполняет разделение строк на сегменты. Размеры сегментов определяются границами строки и экземпляров.
substr(m,n)	Возвращает часть строки, которая начинается с заданной позиции и содержит заданное количество символов. При передаче отрицательного числа отсчет будет выполняться с конца строки.
substring(m,n)	Возвращает часть строки между начальной и конечной позициями.
toLowerCase()	Преобразует символы исходной строки в нижний регистр.
toSource()	Возвращает строковое представление объекта String.
toUpperCase()	Преобразует все символы строки в верхний регистр.

Библиотека БГУИР

length	Возвращает длину строки.
prototype	Позволяет добавлять новые свойства в экземпляры объекта String.

Методы, относящиеся к форматированию

anchor()	Создает экземпляр дескриптора <a> с атрибутом name, принимающим значение строки, передаваемой методу.
big()	Преобразует строку в экземпляр дескриптора <big>.
blink()	Преобразует строку в экземпляр дескриптора <blink>.
bold()	Преобразует строку в экземпляр дескриптора <bold>.
fixed()	Преобразует строку в экземпляр дескриптора <tt>, определяющего шрифт фиксированной ширины.
fontcolor()	Устанавливает атрибут color экземпляра дескриптора <font>.
fontsize()	Устанавливает атрибут size экземпляра дескриптора <font>.
italics()	Преобразует строку в экземпляр дескриптора <i>.
link()	Преобразует строку в экземпляр дескриптора <a> и устанавливает атрибут href в значение URL
small()	Преобразует строку в экземпляр дескриптора <small>.
strike()	Преобразует строку в экземпляр дескриптора <strike>.
sub()	Преобразует строку в экземпляр дескриптора <sub>.
sup()	Преобразует строку в экземпляр дескриптора <sup>.

Кроме того, можно выполнять поиск текста внутри строки, прибегнув к помощи методов **indexOf**(), **lastIndexOf**(). Вызывайте их, если требуется отыскать определенные символы или подстроки внутри строки и возвратить соответствующую позицию (или индекс). В то время как **indexOf**() начинает с левой стороны строки и продвигается вправо, **lastIndexOf**() выполняет аналогичные действия, но в обратном направлении. Оба метода начинают действовать с позиции 0 и возвращают значение -1, если текст не найден.

Библиотека ГГТУ им. П.О.Сухого

Пример использования indexOf

```
<script type=" text/javascript">
<!--
// Объявление переменных
   var pos = 0;
   var num = -1;
   var i = -1;
   var graf = "Деньги не приносят счастья, однако обеспечивают
спокойствие и уверенность." ;
   // Поиск в строке с подсчетом количества
   // встречаемых "е"
   while (pos != -1) {
      pos = graf.indexOf("е", i + 1) ;
      num += 1 ;
      i = pos;
   }
   // Поместить ответ на страницу
   document.write(graf);
   document.write("<hr size='l'>");
   document.write("В параграфе найдено " + num + "букв s.") ;
   document.close();
//-->
</script>
```

Если требуется достичь большего, нежели поиск позиции определенного символа, с помощью метода **substring**() можно получить часть строковой переменной или литерала. **substring**() принимает два параметра - начальную и конечную позиции подстроки. Точно так же как и **indexOf**() и **lastIndexOf**(), данный метод начинает отсчет с нуля, т.е. первая позиция в строке рассматривается как 0-вая.

В следующем коде переменная results получит значение Новая:

```
var myString = new String ("Новая Англия");
var results = myString.substring(0,3);
```

Для получения одиночного символа используйте charAt(). Метод CharAt() возвращает символ по указанной позиции, переданной в

Библиотека ГГТУ им.П.О.Сухого

качестве параметра. В при веденном ниже примере results получит значение *и*:

```
var myString = new String("Слоны и бизоны");
var results = myString.charAt(6);
```

В случае указания позиции, выходящей за пределы строки, возвращается значение *-1*:

```
var myString = new String ("Агент 007);
var results = myString.charAt(20102);
```

6.2. РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

Сравнение по образцу

На сегодняшний день сформировались две наиболее распространенных области использования Web-страниц - это сбор и распространение данных. По самой своей природе две упомянутых области для определения того, что требуется пользователю, применяют обработку данных, а затем возвращают информацию, подобранную в соответствии с требованиями пользователя. Одним из ключевых инструментов обработки данных является сравнение по образцу.

Реализация сравнения по образцу в языке JavaScript в значительной мере способствует успешной обработке данных для целей Internet. Для выполнения сравнения по образцу в JavaScript используется объект **RegExp**. В этом объекте содержится образец и, кроме того, он предоставляет методы для выполнения самого сравнения.

Объект RegExp можно создать двумя различными способами, которые аналогичны способам создания строк. Одного и того же результата можно добиться как спомощью конструктора RegExp, так и с помощью ключевого слова new:

```
var lastName = new RegExp("Jones");
```

Приведенная строка создает объект RegExp с именем lastName и присваивает емуобразец Jones. Того же можно добиться и путем непосредственного присваивания:

```
var lastName = /Jones/;
```

Для того чтобы отличить создание регулярного выражения от создания строки, в качестве маркера начала и окончания образца применяется косая черта (/).

Библиотека ГГТУ им. П.О. Сухого

Создание образцов

Синтаксис, который используется для определения образцов в JavaScript, можно считать самим языком сценариев, поскольку он достаточно обширный. Существуют специальные символы для создания практически любых образцов, включая символы для работы с группами, повторениями, позициями и т.д. Специальные символы, используемые в JavaScript при выполнении сравнения по образцу, перечислены в таблице

Специальные символы, используемые при выполнении сравнения по образцу

Символ	Описание
\w	Совпадение с любым алфавитно-цифровым символом.
\W	Совпадение с любым символом, не являющимся алфавитно-цифровым.
\s	Совпадение с любым служебным символом (табуляция, новая строка, возврат каретки, прогон страницы, вертикальная табуляция).
\S	Совпадение с любым символом, не являющимся служебным.
\d	Совпадение с любой цифрой.
\D	Совпадение с любым символом, не являющимся цифрой.
\b	Совпадение с символом забоя.
.	Совпадение с любым символом кроме символа новой строки.
[...]	Совпадение с любым одним символом, заключенным в скобки.
[^...]	Совпадение с любым одним символом, не заключенным в скобки,
[x-y]	Совпадение с любым символом из диапазона от x до y.
[^x-y]	Совпадение с любви символом, не входящим в диапазон от x до y.
{x,y}	Совпадение с предыдущим элементом не менее x раз и не более y раз.
{x,}	Совпадение с предыдущим элементом не менее x раз.
?	Совпадение с предыдущим элементом не более одного раза.
+	Совпадение с предыдущим элементом хотя бы один раз.
*	Совпадение с предыдущим элементом любое количество

Библиотека ГГТУ им.П.О.Сухого

	раз или ни одного раза.
|	Совпадение с выражением слева или справа от символа |.
(...)	Группировка всего содержимого скобок в подобразец.
\x	Совпадение с символами, которые получены из подобразца в группе номер x. Группы, которые обозначаются скобками, нумеруются слева направо.
^	Установка соответствия с началом строки в многострочных сравнениях.
$	Установка соответствия с концом строки в многострочных сравнениях.
\b	Сравнение позиции между алфавитно-цифровым символом и неалфавитно-цифровым символом.
\B	Сравнение позиции, которая не находится между алфавитно-цифровым символом и неалфавитно-цифровым символом.

Описанные специальные символы сравнения по образцу используются для создания сложных образцов. Из приведенной таблицы можно заметить, что символы, подобные звездочке (*), знаку плюс (+) и обратной косой черте (\), имеют особое назначение ,что не позволяет их интерпретировать как литералы. А что делать, если в строке необходимо найти все знаки плюс (+)? Чтобы использовать знак плюс в качестве литерала, в определении образца ему должна предшествовать обратная косая черта(\).

Все символы, которые для использования их в качестве литералов требуют указания обратной косой черты, перечислены в следующей таблице

\f	Прогон страницы
\n	Новая строка
\r	Возврат каретки
\t	Табуляция
\V	Вертикальная табуляция
\/	Прямая косая черта (/)
\\	Обратная косая черта (\)
\.	Точка (.)
*	Звездочка (*)
\+	Плюс (+)
\?	Знак вопроса (?)
\\|	Вертикальная черта (|)
\(	Открывающая круглая скобка (

Библиотека БГУИР

\)	Закрывающая круглая скобка)
\[	Открывающая квадратная скобка [
\]	Закрывающая квадратная скобка]
\{	Открывающая фигурная скобка {
\)	Закрывающая фигурная скобка }
\xxx	ASCII-символ, соответствующий восьмеричному коду XXX
\xHH	ASCII-символ, соответствующий шестнадцатеричному коду HH
\cX	Управляющий символ, соответствующий коду X

В конструкторе RegExp есть еще необязательный параметр, задающий опции поиска. Вот его параметры.

g	Глобальное сравнение. Находит все возможные совпадения.
m	Многострочный поиск
i	Сравнение, не чувствиетльное к регистру

Проверка на совпадение по образцу

После создания образец можно применить к строке, воспользовавшись специальными методами объектов RegExp и String. Методы сравнения по образцу объекта String, как показано в таблице, требуют объекты RegExp.

match(regExpObj)	Осуществляет в строке поиск образца regExpObj и возвращает результат.
replace(regExpObj,str)	Заменяет все экземпляры образца reqExpObj строкой str.
search(reqExpObj)	Возвращает позицию образца reqExpObj в строке.
split(regExpObj,max)	Разбивает строку по каждому экземпляру образца reqExpObj, но не более max раз. Подстроки возвращаются в виде массива.

Методы сравнения по образцу объекта RegExp, как показано в следующей таблице, требуют объекты String.

exec(str)	Выполняется поиск образца в строке str и возвращается результат
test(str)	Выполняется поиск образца в str и возвращается true, если совпадение найдено и false в

Библиотека ГГТУ им. П.О.Сухого

	противном случае
(str)	Аналогичен методу str

```
<html>
<script type="text/javascript" language="JavaScript">
   //Создание строки
   var str = "Джон обменял 5 апельсинов на 135 ягод винограда.<br>";
   //Создание объекта RegExp
   var span3to5 = new RegExp("[3-5]","g");
   document.write(str) ;
   document.write("Заменить цифры от 3 до 5 цифрой 9.<br>");
   document.write(str.replace(span3to5,"9"));
</script>
</html>
```

В приведенном листинге для замены цифр строки от 3 до 5 цифрой 9 используется конструктор RegExp, специальный синтаксис образца и метод replace() объекта String.

РАЗДЕЛ 7. ОБЪЕКТНАЯ МОДЕЛЬ ДОКУМЕНТА

7.1. ОБЪЕКТ WINDOW

Класс объектов Window — это самый старший класс в иерархии объектов JavaScript. Объект window, относящийся к текущему окну (т.е. в котором выполняется скрипт), является объектом класса Window. Класс объектов Frame содержится в классе Window, т.е. каждый фрейм — это тоже объект класса Window.

У объекта window могут быть разные свойства при загрузке разных страниц. Кроме того, в разных браузерах свойства объектов и поведение объектов и браузера при обработке событий может быть различным. При программировании на JavaScript чаще всего используют следующие свойства, методы и события объекта window:

Свойства	Методы	События
status	open()	Load
defaultStatus	close()	Unload
location	focus()	
history	blur()	Focus

Библиотека БГТУ www.bstu.by

navigator		Blur
	alert()	
document	confirm()	Resize
frames[ ]	prompt()	Error
opener	setTimeout()	
parent	setInterval()	
self	clearTimeout()	
top	clearInterval()	

Поскольку объект window является самым старшим, то в большинстве случаев при обращении к его свойствам и методам приставку "window." можно опускать. Так, например, можно писать alert('Привет') вместо window.alert('Привет'), или location вместо window.location. Исключениями из этого правила являются вызовы методов open() и close(), у которых нужно указывать имя окна, с которым работаем (родительское в первом случае и дочернее во втором). Свойства frames[], self, parent и top будут рассмотрены в разделе, посвященном фреймам. Свойство opener будет рассмотрено при описании метода window.close().

Свойства объекта window

Поле статуса и свойство window.status

Поле статуса — это первое, что начали использовать авторы HTML-страниц из арсенала JavaScript. Бегущая строка в поле статуса была изюминкой, которая могла действительно привлечь внимание пользователей к Web-узлу. Постепенно ее популярность сошла на нет. Бегущие строки стали редкостью, но программирование поля статуса встречается на многих Web-узлах.

Полем статуса (status bar) называют поле нижней части окна браузера сразу под областью отображения HTML-страницы. В поле статуса отображается информация о состоянии браузера (загрузка документа, загрузка графики, завершение загрузки, запуск апплета и т.п.). Программа на JavaScript имеет возможность работать с этим полем как с изменяемым свойством окна. При этом фактически с ним связаны два разных свойства:

window.status — значение поля статуса;

window.defaultStatus — значение поля статуса по умолчанию.

Библиотека БГТУ им. П.О. Сухого

Значение свойства status можно изменить — и оно тут же будет отображено в поле статуса. Свойство defaultStatus тоже можно менять — и сразу по его изменении оно отображается в поле статуса.

Разница между этими двумя свойствами заключается в их поведении: если свойству status присвоить пустую строку: window.status="", то в поле статуса автоматически будет отображено значение defaultStatus. Обратного же не происходит: при присвоении пустой строки свойству defaultStatus оно и отобразится в поле статуса, независимо от значения свойства status. Следует отметить, что реакция браузеров на описываемые ниже действия со свойствами status и defaultStatus может быть разной в различных браузерах.

Программируем status

```
<A onMouseOver="window.status='Мышь над ссылкой';return true;"
   onMouseOut="window.status='Мышь увели со ссылки';"
   HREF="http://site.com/">Наведите мышь на ссылку и следите за полем статуса</A>
```

Обратите внимание на оператор **return true** в конце обработчика событий **onMouseOver**. Он необходим для того, чтобы отменить действие по умолчанию, которое, в отсутствие этого оператора, браузер выполнил бы сразу после вывода нами своей строки в поле статуса, и пользователь не успел бы увидеть нашу строку. Аналогичный трюк отмены действия по умолчанию годится и для некоторых других событий (**onClick, onKeyDown, onKeyPress, onMouseDown, onMouseUp, onSubmit, onReset**), с той лишь разницей, что для перечисленных обработчиков отмена выполняется оператором **return false**.

Для обработчика onMouseOut такого способа отменить действие по умолчанию не Ситуация изменится в следующем примере, когда мы предварительно зададим свое (непустое) значение defaultStatus.

Программируем defaultStatus

Свойство defaultStatus определяет текст, отображаемый в поле статуса, когда никаких событий не происходит. Дополним предыдущий пример изменением этого свойства в момент окончания загрузки документа, т.е. в обработчике onLoad:

Библиотека ГГТУ им.П.О.Сухого

```
<BODY onLoad="window.defaultStatus='Значение по умолчанию';">
<A onMouseOver="window.status='Мышь над ссылкой';return true;"
   onMouseOut="window.status='Мышь увели со ссылки';
    alert('Ждем');"
   HREF="http://site.com/">Наведите мышь на ссылку и следите
   за полем статуса</A>
</BODY>
```

Сразу после загрузки документа в поле статуса будет "Значение по умолчанию". При наведении указателя мыши на ссылку в поле статуса появится надпись "Мышь над ссылкой", при этом URL ссылки (http://site.com/) в поле статуса не появится, т.к. мы подавили его вывод оператором return true.

Поле адреса и свойство window.location

Поле адреса в браузере обычно располагается в верхней части окна и отображает URL загруженного документа. Если пользователь хочет вручную перейти к какой-либо странице (набрать ее URL), он делает это в поле адреса.

Свойство location объекта window само является объектом класса Location. Класс Location, в свою очередь, является подклассом класса URL, к которому относятся также объекты классов Area и Link.

В целях совместимости с прежними версиями JavaScript, в языке поддерживается также свойство **window.document.location**, которое в настоящее время полностью дублирует свойство window.location со всеми его свойствами и методами.

Свойства объекта location

Предположим, что браузер отображает страницу, расположенную по адресу:

http://www.site.ru:80/dir/page.cgi?product=phone&id=3#mark

Тогда свойства объекта location примут следующие значения:

```
window.location.href=
"http://www.site.ru:80/dir/page.cgi?product=phone&id=3#mark"
window.location.protocol = "http:"
```

```
window.location.hostname = "www.site.ru"
window.location.port     = 80
window.location.host     = "www.site.ru:80"
window.location.pathname = "dir/page.cgi"
window.location.search   = "?product=phone&id=3"
window.location.hash     = "#mark"
```

Методы объекта location

Методы объекта location предназначены для управления загрузкой и перезагрузкой страницы. Это управление заключается в том, что можно либо перезагрузить текущий документ (метод **reload()**), либо загрузить новый (метод **replace()**).

```
window.location.reload(true);
```

Метод **reload()** полностью моделирует поведение браузера при нажатии на кнопку Reload в панели инструментов. Если вызывать метод без аргумента или указать его равным true, то браузер проверит время последней модификации документа и загрузит его либо из кеша. Если в качестве аргумента указать false, то браузер перезагрузит текущий документ с сервера, несмотря ни на что

Используя объект location, перейти на новую страницу можно двумя способами:

```
window.location.href="http://www.newsite.ru/";
window.location.replace("http://www.newsite.ru/");
```

Разница между ними — в отображении этого действия в истории посещений страниц window.history. В первом случае в историю посещений добавится новый элемент, содержащий адрес "http://www.newsite.ru/", так что при желании можно будет нажать кнопку Back на панели браузера, чтобы вернуться к прежней странице. Во втором случае новый адрес "http://www.newsite.ru/" заместит прежний в истории посещений, и вернуться к прежней странице нажатием кнопки Back уже будет невозможно.

История посещений (history)

История посещений страниц World Wide Web позволяет пользователю вернуться к странице, которую он просматривал ранее

Библиотека ГГТУ им. П.О. Сухого

в данном окне браузера. История посещений в JavaScript трансформируется в объект window.history. Этот объект указывает на массив URL-страниц.

Чтобы не возникло проблем с безопасностью браузера, путешествовать по History можно, только используя индекс. При этом URL, как текстовая строка, программисту недоступен. Чаще всего этот объект используют в примерах или страницах, на которые могут быть ссылки из нескольких разных страниц, предполагая, что можно вернутся к странице, из которой пример будет загружен:

```
<FORM><INPUT TYPE="button" VALUE="Назад"
    onClick="history.back()"></FORM>
```

Данный код отображает кнопку "Назад", нажав на которую, мы вернемся на предыдущую страницу. Аналогичным образом действует метод history.forward(), перенося нас на следующую посещенную страницу.

Существует также метод **go(),** имеющий целочисленный аргумент и позволяющий перескакивать на несколько шагов вперед или назад по истории посещений. Например, **history.go(-3)** перенесет нас на 3 шага назад в истории просмотра. При этом методы **back()** и **forward()** равносильны методу go() с аргументами -1 и 1, соответственно. Вызов **history.go(0)** приведет к перезагрузке текущей страницы.

Тип браузера (navigator)

Часто возникает задача настройки страницы на конкретную программу просмотра (браузер). При этом возможны два варианта: определение типа браузера на стороне сервера, либо на стороне клиента. Для последнего варианта в арсенале объектов JavaScript существует объект window.navigator. Важнейшие из свойств этого объекта перечислены ниже.

userAgent	Основная информация о браузере. Передается серверу в HTTP-заголовке при открытии пользователем страниц
appName	Название браузера
appCodeName	Кодовое название браузера
appVersion	Данные о версии браузера и совместимости

Рассмотрим простой пример определения типа программы просмотра:

```
<FORM><INPUT TYPE=button VALUE="Тип навигатора"
onClick="alert(window.navigator.userAgent);"></FORM>
```

При нажатии на кнопку отображается окно предупреждения, содержащее значение свойства navigator.userAgent. Если это значение разобрать по компонентам, то может получиться, например, следующее:

```
navigator.userAgent    = "Mozilla/4.0 (compatible; MSIE 7.0; Windows
   NT 6.0; SLCC1)"
navigator.appName     = "Microsoft Internet Explorer"
navigator.appCodeName = "Mozilla"
navigator.appVersion   = "4.0 (compatible; MSIE 7.0; Windows NT 6.0;
   SLCC1)"
```

У объекта navigator есть еще несколько интересных с точки зрения программирования применений. Например, чтобы проверить, поддерживает ли браузер клиента язык Java, достаточно вызвать метод **navigator.javaEnabled**.

Можно проверить, какие форматы графических файлов поддерживает браузер, воспользовавшись свойством navigator.mimeTypes (оно представляет собой массив всех типов MIME, которые поддерживаются данным браузером):

```
<SCRIPT>
if(navigator.mimeTypes['image/gif']!=null)
  document.write('Ваш браузер поддерживает GIF<BR>');

if(navigator.mimeTypes['image/tif']==null)
  document.write('Ваш браузер не поддерживает TIFF');
</SCRIPT>
```

window.open()

Метод open() предназначен для создания новых окон. В общем случае его синтаксис выглядит следующим образом:

```
myWin =
        window.open("URL", "имя_окна",
        "параметр=значение,параметр=значение,...",
        заменить);
```

Первый аргумент задает адрес страницы, загружаемой в новое окно (можно оставить пустую строку, тогда окно останется пустым).

Второй аргумент задает имя окна, которое можно будет использовать в атрибуте TARGET контейнеров <A> и <FORM>. В

Библиотека ГГТУ им. П.О.Сухого

качестве значений допустимы также зарезервированные имена _blank, _parent, _self, _top, смысл которых такой же, как у аналогичных значений атрибута TARGET. Если имя_окна совпадает с именем уже существующего окна (или фрейма), то новое окно не создается, а все последующие манипуляции с переменной myWin будут применяться к этому окну (или фрейму).

Третий аргумент есть *не содержащая пробелов* строка, представляющая собой список параметров и их значений, перечисленных через запятую. Указание каждого из параметров необязательно, однако значения по умолчанию могут зависеть от браузера, поэтому всегда указывайте явно те параметры, на которые рассчитываете. Возможные параметры перечислены в таблице. Вместо значений yes и no можно использовать 1 и 0.

Последний аргумент "заменить" является необязательным, принимает значения true и false и означает: следует ли новый URL добавить в history в качестве нового элемента или заменить им последний элемент history.

Метод window.open() возвращает ссылку на вновь открытое окно, т.е. объект класса Window. Его можно присвоить переменной (что мы и сделали выше), с тем чтобы потом можно было управлять открытым окном (писать в него, читать из него, передавать и убирать фокус, закрывать).

Параметры метода window.open()

Параметр	Значения	Описание
width	число	Ширина окна в пикселах (не менее 100)
height	число	Высота окна в пикселах (не менее 100)
left	число	Расстояние от левого края экрана до левой границы окна в пикселах
top	число	Расстояние от верхнего края экрана до верхней границы окна в пикселах
directories	yes/no	Наличие у окна панели папок (Netscape Navigator)
location	yes/no	Наличие у окна поля адреса
menubar	yes/no	Наличие у окна панели меню
resizable	yes/no	Сможет ли пользователь менять размер окна
scrollbars	yes/no	Наличие у окна полос прокрутки
status	yes/no	Наличие у окна поля статуса
toolbar	yes/no	Наличие у окна панели инструментов

Библиотека БГУИР

Приведем два примера открытия нового окна:

```
<FORM>
<INPUT TYPE=button VALUE="Простое окно"
 onClick="window.open('', 'test1',
  'directories=no,height=200,location=no,'+
  'menubar=no,resizable=no,scrollbars=no,'+
  'status=no,toolbar=no,width=200');">

<INPUT TYPE=button VALUE="Сложное окно"
 onClick="window.open('', 'test2',
  'directories=yes,height=200,location=yes,'+
  'menubar=yes,resizable=yes,scrollbars=yes,'+
  'status=yes,toolbar=yes,width=200');">
</FORM>
```

window.close()

Метод close() позволяет закрыть окно. Если необходимо закрыть текущее, то:

```
window.close();
self.close();
```

Если мы открыли окно с помощью метода window.open(), то из скрипта, работающего в новом окне, сослаться на окно-родитель можно с помощью window. Поэтому, если необходимо закрыть родительское окно, т.е. окно, из которого было открыто текущее, то:

```
window.opener.close();
```

Если необходимо закрыть произвольное окно, то тогда сначала нужно получить его идентификатор:

```
id=window.open();
...
id.close();
```

Методы focus() и blur()

Метод focus() применяется для передачи фокуса в окно, с которым он использовался. Передача фокуса полезна как при открытии окна, так и при его закрытии, не говоря уже о случаях, когда нужно выбирать окна. Рассмотрим пример.

```
<HTML><HEAD>
<SCRIPT>
```


Библиотека ГГТУ им.П.О.Сухого

```
function myfocus(a)
{
   myWin = window.open('','example','width=300,height=200');
   // открываем окно и заводим переменную с указателем на него.
   // Если окно с именем 'example' существует, то новое окно не создается,
   // а открывается поток для записи в имеющееся окно с именем 'example'
   if(a==1){
      myWin.document.open(); //открываем поток ввода в уже
созданное окно
      myWin.document.write('<H1>Открыли окно в первый раз');
//Пишем в этот поток
   }
   if(a==2){
      myWin.document.open();
      myWin.document.write('<H1>Открыли окно во второй раз');
   }
   if(a==3){
      myWin.focus(); // передаем фокус, а затем выполняем те же
действия,
      // что и в предыдущем случае
      myWin.document.open();
      myWin.document.write('<H1>Открыли окно в третий раз');
   }
   myWin.document.write('</H1>');
   myWin.document.close();
}</SCRIPT>
</HEAD>
<BODY>
<a href="javascript:myfocus(1);">Откроем окно и напишем в него что-
то</a>,
<BR><BR>
<a href="javascript:myfocus(2);">напишем в него же что-то другое, но
фокус не передадим</a> ,
<BR><BR>
<a href="javascript:myfocus(3);">опять что-то напишем в него, но
сперва передав ему фокус</a>.
</BODY>
</HTML>
```


Библиотека БГУИР

Чтобы увести фокус из определенного окна myWin, необходимо применить метод **myWin.blur().** Например, чтобы увести фокус с текущего окна, где выполняется скрипт, нужно вызвать window.blur(). Эффект будет тот же, как если бы пользователь сам свернул окно нажатием кнопки в правом верхнем углу окна.

Метод setTimeout()

Метод setTimeout() используется для создания нового потока вычислений, исполнение которого откладывается на время (в миллисекундах), указанное вторым аргументом:

idt = setTimeout("JavaScript_код",Time);

Типичное применение этой функции — организация периодического изменения свойств объектов. Например, можно запустить часы в поле формы:

```
<HTML><HEAD><SCRIPT>
var Chasy_idut=false;

function myclock(){
   if(Chasy_idut){
      d = new Date();
      document.f.c.value =
      d.getHours()+':'+
      d.getMinutes()+':'+
      d.getSeconds();
   }
   setTimeout("myclock();",500);
}
function FlipFlag(){
   Chasy_idut = !Chasy_idut;
   document.f.b.value = (Chasy_idut) ? 'Остановить' : 'Запустить';
}
</SCRIPT></HEAD>
<BODY onLoad="myclock();">
<FORM NAME=f>Текущее время:<INPUT NAME=c size=8>
<INPUT TYPE=button name=b
VALUE="Запустить"onClick="FlipFlag();">
</FORM></BODY></HTML>
```


Библиотека ГГТУ им. П.О. Сухого

Метод clearTimeout()

Метод clearTimeout() позволяет уничтожить поток, вызванный методом setTimeout(). Очевидно, что его применение позволяет более эффективно распределять ресурсы вычислительной установки. Для того чтобы использовать этот метод в примере с часами, нам нужно модифицировать функции и форму:

```
<HTML><HEAD><SCRIPT>
   var Chasy_idut=false;
   var potok;

   function StartClock(){
      d = new Date();
      document.f.c.value =
      d.getHours()+':'+
      d.getMinutes()+':'+
      d.getSeconds();
      potok = setTimeout('StartClock();',500);
      Chasy_idut=true;
   }

   function StopClock(){
      clearTimeout(potok);
      Chasy_idut=false;
   }
</SCRIPT></HEAD><BODY>
<FORM NAME=f>
Текущее время:<INPUT NAME=c size=8>
<INPUT TYPE=button VALUE="Запустить" onClick="if(!Chasy_idut)
StartClock();">
<INPUT TYPE=button VALUE="Остановить" onClick="if(Chasy_idut)
StopClock();">
</FORM></BODY></HTML>
```

Методы setInterval() и clearInterval()

В предыдущих примерах для того, чтобы поток запускался снова и снова, мы помещали в функцию в качестве последнего оператора вызов метода setTimeout(). Однако в JavaScript для этих целей имеются специальные методы. Метод **setInterval("код_JavaScript",time)** выполняет код_JavaScript с

Библиотека БГТУ им. П.О. Сухого

периодом раз в time миллисекунд. Возвращаемое значение — ссылка на созданный поток. Чтобы остановить поток, необходимо вызвать метод **clearInterval(поток)**.

События объекта window

Остановимся вкратце на событиях, связанных с объектом window. Обработчики этих событий обычно помещают как атрибут контейнера <BODY>.

onLoad — событие происходит в момент, когда загрузка документа в данном окне полностью закончилась. Если текущим окном является фрейм, то событие Load его объекта window происходит, когда в данном фрейме загрузка документа закончилась, независимо от состояния загрузки документов в других фреймах. Использовать обработчик данного события можно, например, следующим образом:

```
<BODY onLoad="alert('Документ полностью загружен.');">
```

onUnload — событие происходит в момент выгрузки страницы из окна. Например, когда пользователь закрывает окно, либо переходит с данной Web-страницы на другую, кликнув ссылку или набрав адрес в адресной строке, либо при изменении адреса страницы (свойства window.location) скриптом. Например, при уходе пользователя с нашей страницы мы можем позаботиться о его удобстве и закрыть открытое ранее нашим скриптом окно:

```
<BODY onUnload="myWin.close();">
```

onError — событие происходит при возникновении ошибки в процессе загрузки страницы. Если это событие произошло, можно, например, вывести сообщение пользователю с помощью alert() или попытаться перезагрузить страницу с помощью window.location.reload

onFocus — событие происходит в момент, когда окну передается фокус. Например, когда пользователь "раскрывает" свернутое ранее окно, либо (в Windows) выбирает это окно браузера с помощью Alt+Tab среди окон других приложений. Это событие происходит также при программной передаче фокуса данному окну путем вызова метода window.focus(). Пример использования:

```
<BODY onFocus="alert('Спасибо, что снова вернулись!');">
```

onBlur — событие, противоположное предыдущему, происходит в момент, когда данное окно теряет фокус. Это может произойти в результате действий пользователя либо программными средствами — вызовом метода window.blur().

Библиотека ГГТУ им. П.О.Сухого

onResize — событие происходит при изменении размеров окна пользователем либо сценарием.

Переменные как свойства окна

Глобальные переменные на самом деле являются свойствами объекта window. В следующем примере мы открываем окно с идентификатором wid, заводим в нем глобальную переменную t и затем пользуемся ею в окне-родителе, ссылаясь на нее как wid.t:

```
<HTML><HEAD>
<SCRIPT>
   wid = window.open('','','width=750,height=100,status=yes');
   wid.document.open(); R = wid.document.write;
   R('<HTML><HEAD><SCRIPT>var t;<\/SCRIPT></HEAD>');
   R('<BODY><H1>Новое окно</H1></BODY></HTML>');
   wid.document.close();
</SCRIPT>
</HEAD>
<BODY>
<A HREF="javascript:wid.t=window.prompt('Новое состояние:','');
wid.status=wid.t; wid.focus(); void(0);">Изменим значение переменной
t в новом окне</A>
</BODY></HTML>
```

Аналогичным образом (с приставкой wid, указывающей на объект окна) можно обращаться ко всем элементам, находящимся в открытом нами окне, например, к формам. В качестве примера рассмотрим изменение поля ввода в окне-потомке из окна-предка. Создадим дочернее окно с помощью функции okno(), в нем создадим форму, а затем обратимся к полю этой формы из окна-предка:

```
<HTML><HEAD>
<SCRIPT>
var wid; // Объявляем глобальную переменную
function okno()
{
   wid = window.open('','okoshko','width=500,height=200');
   wid.document.open(); R = wid.document.write;
   R('<HTML><BODY><H1>Меняем текст в окне-потомке:</H1>');
   R('<FORM NAME=f><INPUT SIZE=40 NAME=t VALUE=Текст>');
   R('</FORM></BODY></HTML>');
   wid.document.close();
```

```
}
</SCRIPT>
</HEAD><BODY>
<INPUT TYPE=button VALUE="Открыть окно примера"
onClick="okno()">
<INPUT TYPE=button VALUE="Написать текущее время в поле
ввода"
      onClick="window.wid.document.f.t.value=new Date();
              window.wid.focus();">
</BODY>
</HTML>
```

Переменная wid должна быть глобальной, т.е. определена за пределами каких-либо функций (как сделано в нашем примере). В этом случае она становится свойством объекта window, поэтому мы обращаемся к ней в обработчике onClick посредством window.wid.

7.2. ОБЪЕКТ DOCUMENT

Объект document является важнейшим свойством объекта window (т.е. полностью к нему нужно обращаться как window.document). Все элементы HTML-разметки, присутствующие на web-странице, — текст, абзацы, гиперссылки, картинки, списки, таблицы, формы и т.д. — являются свойствами объекта document. Можно сказать, что технология DHTML (Dynamic HTML), т.е. динамическое изменение содержимого web-страницы, заключается именно в работе со свойствами, методами и событиями объекта document (не считая работы с окнами и фреймами).

Свойства	Методы	События
URL	open()	Load
domain	close()	Unload
title		
lastModified	write()	Click
referrer	writeln()	DblClick
cookie		
	getSelection()	MouseDown
linkColor		MouseUp
alinkColor	getElementById()	
vlinkColor	getElementsByName()	KeyDown

Библиотека БГУИР

	getElementsByTagName()	KeyUp KeyPress

С одним методом мы уже часто работали: **document.write()** — он пишет в текущий HTML-документ. Его модификация **document.writeln()** делает то же самое, но дополнительно добавляет в конце символ новой строки; это удобно, если потом требуется читать сгенерированный HTML-документ глазами. Если запись идет в HTML-документ нового окна, открытого с помощью **window.open(),** то перед записью в него нужно открыть поток на запись с помощью метода **document.open(),** а по окончании записи закрыть поток методом **document.close().** После выполнения последнего действия произойдет событие **onLoad** (и вызовется соответствующий обработчик события **onLoad**) у документа, а затем у окна.

События объекта document аналогичны одноименным событиям объекта window (только у document они происходят раньше), либо их смысл понятен из их названия, поэтому мы не будем детально их разбирать.

Остановимся вкратце на свойствах объекта document. Свойства **linkColor, alinkColor и vlinkColor** задают цвет гиперссылок — непосещенных, активных и посещенных, соответственно.

Свойство **URL** хранит адрес текущего документа (т.е. строковый литерал, равный window.location.href, если страница состоит из единственного документа, а не является набором фреймов). Свойство **domain** выдает. Свойство **title** выдает заголовок страницы (указанный в контейнере <TITLE>), **lastModified** указывает на дату и время последней модификации файла, в котором содержится данный HTML-документ (без учета времени модификации внешних файлов — стилевых, скриптов и т.п.). Свойство referrer выдает адрес страницы, с которой пользователь пришел на данную web-страницу, кликнув по гиперссылке. Наконец, свойству **cookie** посвящен целый раздел в лекции 8.

Фреймы (Frames)

Фреймы — это несколько видоизмененные окна. Отличаются они от обычных окон тем, что размещаются внутри них. У фрейма не может быть ни панели инструментов, ни меню, как в обычном окне. В качестве поля статуса фрейм использует поле статуса окна, в котором он размещен. Существует и ряд других отличий.

Библиотека БГУИР

Если окно имеет фреймовую структуру (т.е. вместо контейнера <BODY> в нем присутствует контейнер <FRAMESET> со вложенными в него контейнерами <FRAME> и быть может другими контейнерами <FRAMESET>), то объект window соответствует внешнему контейнеру <FRAMESET>, а с каждым вложенным контейнером <FRAME> ассоциирован свой собственный объект класса Window.

Каждому окну или фрейму создатель страницы может дать имя — с помощью атрибута NAME контейнера FRAME, либо вторым аргументом метода window.open(). Используется оно в качестве значения атрибута TARGET контейнеров A и FORM, чтобы открыть ссылку или отобразить результаты работы формы в определенном окне или фрейме. Есть несколько зарезервированных имен окон: _self (имя текущего окна или фрейма, где исполняется скрипт), _blank (новое окно), _parent (окно-родитель для данного фрейма), _top (самый старший предок данного фрейма, т.е. окно браузера, частью которого является данный фрейм). Иерархия фреймов, обсуждаемая ниже, как раз и задает, какие окна или фреймы являются родителями для других фреймов.

У каждого объекта класса Window, будь то окно или фрейм, есть также ссылка на соответствующий объект. Как мы знаем, ссылкой на объект текущего окна, в котором исполняется скрипт, является window; кроме того, на него же ссылается свойство self объекта window (а также свойство window объекта window — есть и такое!). Ссылку на объект окна, открываемого методом window.open(), выдает сам этот метод. Ссылка на объект-фрейм совпадает с его именем, заданным с помощью атрибута NAME контейнера FRAME. Наконец, у объектов-фреймов есть специальные свойства, дающие ссылки на родительский фрейм (window.parent) и на окно браузера, частью которого является данный фрейм (window.top).

Иерархия и именование фреймов

Рассмотрим сначала простой пример. Разделим экран на две вертикальные колонки:

```
<HTML>
<HEAD>
<TITLE>Левый и правый</TITLE>
</HEAD>
 <FRAMESET COLS="50%,*">
```


Библиотека БГУИР / ...

```
    <FRAME NAME=leftframe SRC=left.htm>
    <FRAME NAME=rightframe SRC=right.htm>
  </FRAMESET>
</HTML>
```

Усложним пример: разобьем правый фрейм на два по горизонтали:

```
<HTML>
<HEAD>
<TITLE>Левый, верх и низ</TITLE>
</HEAD>
  <FRAMESET COLS="50%,*">
   <FRAME NAME=leftframe SRC=left.htm>
   <FRAMESET ROWS="50%,*">
     <FRAME NAME=topframe SRC=top.htm>
     <FRAME NAME=botframe SRC=bottom.htm>
   </FRAMESET>
  </FRAMESET>
</HTML>
```

Следовательно, мы можем поместить в контейнер <HEAD> следующий скрипт, устанавливающий цвет фона для всех трех фреймов: (открыть)

```
<SCRIPT>
window.onload=f;
function f()
{
 window.leftframe.document.bgColor='blue';
 window.topframe.document.bgColor='red';
 window.botframe.document.bgColor='green';
}
</SCRIPT>
```

Для того чтобы фрейм rightframe все же появился в иерархии и ему подчинялись два правых фрейма, нужно свести оба наших примера в один. Это значит, что во фрейм rightframe мы должны загрузить отдельный фреймовый документ.

```
<HTML>
<HEAD>
</HEAD>
  <FRAMESET COLS="50%,*">
```

```
  <FRAME NAME=leftframe SRC=left.htm>
  <FRAME NAME=rightframe SRC=right.htm>
 </FRAMESET>
</HTML>    <HTML>
<HEAD>
</HEAD>
 <FRAMESET ROWS="50%,*">
   <FRAME NAME=topframe SRC=top.htm>
   <FRAME NAME=botframe SRC=bottom.htm>
 </FRAMESET>
</HTML>
```

Теперь чтобы из главного окна обратиться ко всем трем фреймам и установить в них те же цвета фона, следует писать:

```
 <SCRIPT>
window.onload=f;
function f()
{
   window.leftframe.document.bgColor='blue';
   window.rightframe.topframe.document.bgColor='red';
   window.rightframe.botframe.document.bgColor='green';
}
</SCRIPT>
```

Коллекция фреймов

Выше мы обращались к фрейму по его имени. Однако, если имя не известно (или не задано), либо если нужно обратиться ко всем дочерним фреймам по очереди, то более удобным будет обращение через коллекцию фреймов **frames[],** которая является свойством объекта window.

В качестве иллюстрации предположим, что в примере из двух правый фрейм содержит несколько изображений, и нам требуется поменять адрес (значение атрибута SRC) третьего изображения с помощью скрипта, находящегося в левом фрейме. Правый фрейм — второй, значит, его номер 1; третье изображение имеет номер 2. Поэтому, это можно сделать следующими способами:

```
top.frames[1].document.images[2].src = 'pic.gif';
top.frames['rightframe'].document.images[2].src = 'pic.gif';
top.frames.rightframe.document.images[2].src = 'pic.gif';
top.rightframe.document.images[2].src = 'pic.gif';
```


Библиотека ГГТУ им.П.О.Сухого

РАЗДЕЛ 8. JAVASCRIPT И HTML-ФОРМЫ

8.1. КОНТЕЙНЕР FORM

Если рассматривать программирование на JavaScript в исторической перспективе, то первыми объектами, для которых были разработаны методы и свойства, стали поля форм. Обычно контейнер FORM и поля форм именованы:

```
<FORM NAME=fname METHOD=get>
<INPUT NAME=iname SIZE=30 MAXLENGTH=30>
</FORM>
```

Поэтому в программах на JavaScript к ним обращаются по имени:

```
document.fname.iname.value="Текст";
```

Того же эффекта можно достичь, используя коллекции форм и элементов, обращаясь к форме и к элементу либо по индексу, либо по имени:

```
document.forms[0].elements[0].value="Текст";
document.forms['fname'].elements['iname'].value="Текст";
```

Рассмотрим подробнее объект Form, который соответствует контейнеру FORM.

Свойства	Методы	События
length **action** **method** **target** **encoding** **elements[]**	**reset()** **submit()**	**Reset** **Submit**

Свойство action

Свойство action отвечает за вызов CGI-скрипта. В нем указывается URL этого скрипта. Но там, где можно указать URL, можно указать и его схему javascript:, например:

```
<FORM METHOD=post ACTION="javascript:
alert('Работает!');">
```


Библиотека ГГТУ им.П.О.Сухого

```
<INPUT TYPE=submit VALUE="Продемонстрировать JavaScript в ACTION">
</FORM>
```

Обратите внимание на тот факт, что в контейнере FORM указан атрибут METHOD. В данном случае это сделано для того, чтобы к URL, заданному в атрибуте ACTION, не дописывался символ "?". Дело в том, что методом доступа по умолчанию является метод GET. В этом методе при обращении к ресурсу из формы создается элемент URL под названием search. Этот элемент предваряется символом "?", который дописывается в конец URL скрипта. В нашем случае это привело бы к неправильной работе JavaScript-кода, поскольку конструкция вида **alert('Строка');?** провоцирует ошибку JavaScript. Метод POST передает данные формы скрипту в теле HTTP-сообщения, поэтому символ "?" не добавляется к URL, и ошибка не генерируется. При этом применение void(0) отменяет перезагрузку документа, и браузер не генерирует событие Submit, т.е. не обращается к серверу при нажатии на кнопку, как это было бы при стандартной обработке формы.

Свойство method

Свойство **method** определяет метод доступа к ресурсам HTTP-сервера из программы-браузера. В зависимости от того, как автор HTML-страницы собирается получать и обрабатывать данные из формы, он может выбрать тот или иной метод доступа. На практике чаще всего используются методы GET и POST.

JavaScript-программа может изменить значение этого свойства. В предыдущем разделе метод доступа в форме был указан явно. Теперь мы его переопределим в момент исполнения программы:

```
<FORM NAME=f ACTION="javascript: alert('Работает!');">
<SCRIPT>
document.write('По умолчанию установлен метод:'+document.f.method+'.<BR>');
</SCRIPT>
<INPUT TYPE=button onClick="document.f.method='post'"
VALUE="Сменить метод на POST">
<INPUT TYPE=button onClick="document.f.method='get'"
VALUE="Сменить метод на GET"><BR>
<INPUT TYPE=submit VALUE="JavaScript в ACTION">
</FORM>
```

По умолчанию установлен метод GET.

В тело документа через контейнер SCRIPT встроен JavaScript-код, который сообщает метод доступа, установленный в форме по умолчанию. Этот контейнер расположен сразу за контейнером FORM. Ставить его перед контейнером FORM нельзя, так как в момент получения интерпретатором управления объект FORM не будет создан, и, следовательно, работать с его свойствами не представляется возможным.

Свойство target

Свойство **target** определяет имя окна, в которое следует загружать результат обращения к CGI-скрипту. При этом всегда есть альтернативы: можно использовать значение этого свойства внутри JavaScript-программ для указания окна или фрейма, куда требуется загружать результат работы CGI-скрипта.

Свойство encoding

Свойство **encoding** объекта Form (а также атрибут enctype контейнера FORM) задает, каким образом данные из формы должны быть закодированы перед их отправкой на сервер. Возможные значения:

application/x-www-form-urlencoded	Это значение по умолчанию. Означает, что в данных, передаваемых на сервер, пробелы заменяются на "+", а специальные символы заменяются на их 16-ричное ASCII значение, например, буква Щ заменяется на %D0%A9.
text/plain	Пробелы заменяются на "+", но специальные символы не кодируются (передаются как есть).
multipart/form-data	Никакие символы не кодируются (они передаются как есть). Данное значение необходимо указывать, если в форме имеются элементы отправки файлов: <INPUT TYPE=file>.

Коллекция elements[ ]

При генерации встроенного в документ объекта Form браузер создает и связанный с ним массив (коллекцию) полей формы elements[]. Обычно к полям обращаются по имени, но можно обращаться и по индексу массива полей формы:

Библиотека БГТУ имени П.О. Сухого

```
<FORM NAME=f>
<INPUT NAME=e SIZE=40>
<BR><INPUT TYPE=button VALUE="Ввести текст по имени
элемента"
 onClick="document.f.e.value='Текст введен по имени
элемента';">
<BR><INPUT TYPE=button VALUE="Ввести текст по индексу
элемента"
 onClick="document.f.elements[0].value='Текст введен по
индексу
 элемента';">
<BR><INPUT TYPE=reset VALUE="Очистить">
</FORM>
```

Индексирование полей в массиве начинается с нуля. Общее число полей в форме f доступно двумя способами: как свойство массива document.f.elements.length и как свойство объекта формы: document.f.length.

Методы объекта Form

Метод submit()

Метод **submit()** позволяет проинициировать передачу введенных в форму данных на сервер:

```
<FORM NAME=f ACTION="http://www.intuit.ru/rating_students/">
Ваше имя пользователя на intuit:<INPUT NAME=query>
</FORM>
<A HREF="javascript:document.f.submit();">Посмотреть
рейтинг</A>
```

Как видите, кнопки отправки (submit) у формы нет, но нажав на ссылку, мы выполняем отправку данных на сервер. Обычно при такой "скрытой" отправке данных на сервер браузеры, в целях безопасности, запрашивают подтверждение, действительно ли пользователь желает отправить данные. Отправка данных путем вызова метода submit() имеет отличия от нажатия пользователем кнопки INPUT типа TYPE=submit; их мы рассмотрим в конце лекции.

Метод reset()

Метод reset() (не путать с обработчиком события onReset, рассматриваемым ниже) позволяет восстановить значения полей формы, заданные по умолчанию. Другими словами, вызов метода

Библиотека ГГТУ им. П.О.Сухого

reset() равносилен нажатию на кнопку INPUT типа **TYPE=reset, но при этом саму эту кнопку создавать не требуется.**

```
<FORM NAME=f>
<INPUT VALUE="Значение по умолчанию" SIZE=30>
<INPUT TYPE=button VALUE="Изменим текст в поле ввода"
    onClick="document.f.elements[0].value='Изменили текст';">
</FORM>
<A HREF="javascript:document.f.reset();void(0);">
  Установили значение по умолчанию</A>
```

В данном примере если кликнуть по гипертекстовой ссылке, то в форме происходит восстановление значений полей по умолчанию.

События объекта Form

Событие Submit

Событие Submit возникает (и соответствующий обработчик события onSubmit вызывается) при нажатии пользователем на кнопку типа submit или при выполнении метода submit(). Действие по умолчанию, которое выполняет браузер при возникновении этого события — отправка введенных в поля формы данных на сервер.

Фукцию обработки этого события можно переопределить и даже вовсе отменить. Для этой цели введен атрибут **onSubmit="код_программы"** у контейнера <FORM>. В нем можно указать действия (JavaScript-код), какие должны выполняться при возникновении этого события. Порядок выполнения этих действий и действий браузера, а также использование оператора return false для отмены последних, полностью аналогичны тем, что описаны ниже для onReset. Пример:

```
<SCRIPT>
function TestBeforeSend(){
 if(document.f.query.value=='') {
  alert('Пустую строку не принимаем!');
  return false;
 }
 else return true;
}
</SCRIPT>
<FORM NAME=f METHOD=post onSubmit="return TestBeforeSend();"
    ACTION="http://www.intuit.ru/rating_students/">
Ваше имя пользователя на intuit:<INPUT NAME=query>
```


Библиотека БНТУ им. П.О.Сухого

```
<INPUT TYPE=submit VALUE="Посмотреть рейтинг">
</FORM>
```

Событие Reset

Событие **Reset** возникает (и соответствующий обработчик события onReset вызывается) при нажатии пользователем на кнопку типа reset или при выполнении метода reset(). Действие по умолчанию, которое выполняет браузер при возникновении этого события — восстановление значений по умолчанию в полях формы. Однако функцию обработки этого события можно переопределить и даже вовсе отменить. Для этой цели введен атрибут onReset="код_программы" у контейнера <FORM>. В нем можно указать действия (JavaScript-код), какие должны выполняться при возникновении этого события. Браузер сначала выполняет эти действия, а затем — свое действие по умолчанию. Но если последним оператором в обработчике onReset будет return false, то действие браузера по умолчанию выполняться не будет. Этот прием называется перехватом события. Пример:

```
<FORM onReset="javascript: alert('Не дадим восстановить!');
return false;">
<INPUT VALUE="Измените этот текст" SIZE=30>
<INPUT TYPE=reset VALUE="Восстановить">
</FORM>
```

Здесь команда return false предотвратила восстановление значения поля. Команда return true, равно как и отсутствие оператора return, позволило бы браузеру продолжить обработку события — и восстановить значение поля.

Поля формы и их объекты

У всех объектов, отвечающих полям формы, есть несколько стандартных свойств, доступных только для чтения: **name** (имя элемента, заданное в атрибуте NAME), **type** (тип элемента, например, для контейнеров <INPUT TYPE="..."> он совпадает со значением атрибута TYPE), **form** (указывает на форму f, в которой данный элемент содержится).

При программировании форм часто требуется писать обработчики событий для форм или их элементов, при этом нужно ссылаться на свойства данного элемента, других элементов и формы в целом.

Библиотека БГУИР

Стандартная схема именования по именам либо по индексам обсуждалась выше:

```
document.форма.элемент.свойство     // точечная нотация
document.форма.элемент["свойство"]   // скобочная нотация
document.forms["имя_формы"].elements["имя_элемента"].свойство
document.forms[индекс_формы].elements[индекс_элемента].свойство
```

Однако получающиеся выражения — довольно громоздкие. Поэтому было введено следующее соглашение: в обработчике события формы или элемента формы имя текущего элемента можно опускать (вместе со всей предшествующей "приставкой"). Кроме того, ссылаться на сам текущий элемент можно с помощью ключевого слова this. Последнее может потребоваться, например, когда нужно передать в функцию не какое-то свойство объекта, а сам объект.

```
value  // короче не бывает!
this.value  // здесь this ссылается на элемент "e"
form.e.value  // form есть свойство объекта "e" (равное "f")
this.form.e.value  // комбинируем оба способа
document.f.e.value  // почти полная запись
window.document.f.e.value  // это самая полная запись
document.f.e.form.e.value  // можно итерировать "form.e."
```

8.2. ТЕКСТОВОЕ ПОЛЕ ВВОДА (ОБЪЕКТ TEXT)

Поля ввода (контейнер INPUT типа TYPE=text) являются одним из наиболее популярных объектов программирования на JavaScript. Это объясняется тем, что, помимо использования по прямому назначению, их применяют и в целях отладки программ, выводя в эти поля промежуточные значения переменных и свойств объектов.

```
<A HREF="http://site.com/">ссылка 1</A>
<FORM>Число гипертекстовых ссылок к данному моменту:
<SCRIPT>
	document.write('<INPUT NAME=t
VALUE='+document.links.length+'>');
</SCRIPT>
<BR><INPUT TYPE=button
```

```
VALUE="Число ссылок по окончании загрузки страницы"
onClick="form.t.value=document.links.length;">
<BR><INPUT TYPE=reset>
</FORM>
<A HREF="http://rite.com/">ссылка 2</A>
```

Свойства	Методы	Обработчики событий	
defaultValue **value** **size** **maxLength** **disabled** **readOnly**	**focus()** **blur()** **select()**	**onChange** **onSelect** **onFocus** **onBlur** **onClick** **onDblClick**	**onMouseOver** **onMouseOut** **onMouseDown** **onMouseUp** **onKeyPress** **onKeyDown** **onKeyUp**

Все перечисленные свойства можно менять. Смысл их таков: **value** (текущее значение поля ввода), **defaultValue** (значение поля ввода по умолчанию), **size** (число умещающихся в поле символов, т.е. видимых) **maxLength** (максимальное число символов, которое можно присвоить значению данного поля) **readOnly** (может ли пользователь менять значение поля) **disabled** (может ли пользователь установить фокус на этом поле).

Опишем вкратце методы: **focus()** — устанавливает фокус на данном поле, **blur()** — убирает фокус с данного поля, **select()** — выделяет весь введенный текст (чтобы, например, его можно было скопировать в буфер, либо удалить, нажав клавишу Delete).

Смысл обработчиков событий вполне понятен из их названий: обработчик onChange вызывается, когда пользователь (но не скрипт) изменил значение в поле ввода (и кликнул вне поля ввода); остальные названия очевидны.

8.3. СПИСКИ ВАРИАНТОВ (ОБЪЕКТЫ SELECT И OPTION)

Одним из важных элементов интерфейса пользователя являются списки вариантов. В HTML-формах для их реализации используется контейнер <SELECT>, который вмещает в себя контейнеры <OPTION>. При этом список может "выпадать" либо прокручиваться внутри окна. В зависимости от наличия атрибута MULTIPLE у контейнера <SELECT> список может быть либо с возможностью выбора только одного варианта, либо нескольких вариантов.

Библиотека БГУИР

С каждым контейнером <SELECT> ассоциирован объект класса **Select**, а с каждым дочерним контейнером <OPTION> — объект класса Option, являющийся свойством данного объекта класса Select. Кроме того, свойством объекта класса Select является также коллекция **options[ ]**, объединяющая все его дочерние объекты Option. Перечислим основные свойства, методы и события, характеризующие эти объекты.

Объект Select

Свойства	Методы	Обработчики событий
options[ ] **size** **length** **multiple** **selectedIndex**	**focus()** **blur()** **add()** **remove()**	**onBlur** **onChange** **onFocus**

Объект Option

Свойства	Методы	Обработчики событий
defaultSelected **selected** **index** **text** **value**	**нет**	**нет**

Создание объектов Option

Объект класса Option интересен тем, что в отличие от многих других встроенных в DOM объектов JavaScript, имеет конструктор. Это означает, что программист может сам создать объект класса Option:

opt = new Option([text, [value, [defaultSelected, [selected]]]]);

где аргументы соответствуют свойствам обычных объектов класса Option:

text — строка текста, которая размещается в контейнере <OPTION> (например: <OPTION>текст</OPTION>);

value — значение, которое передается серверу при выборе альтернативы, связанной с объектом Option;

defaultSelected — выбрана ли эта альтернатива по умолчанию (true/false);

selected — альтернатива была выбрана пользователем (true/false).

Библиотека ГГТУ им.П.О.Сухого

На первый взгляд не очень понятно, для чего может понадобиться программисту такой объект, ведь создать объект класса Select нельзя и, следовательно, нельзя приписать ему новый объект Option. Все объясняется, когда речь заходит об изменении списка альтернатив у имеющегося в документе объекта Select. Делать это можно, при этом изменение списка альтернатив Select не приводит к переформатированию документа.

При программировании альтернатив следует обратить внимание на то, что у объектов класса Option нет свойства name, в виду того, что у контейнера <OPTION> нет атрибута NAME. Таким образом, к встроенным в документ объектам класса Option можно обращаться только как к элементам коллекции options[].

Коллекция options[ ]

Встроенный массив (коллекция) options[] — это одно из свойств объекта Select. Элементы этого массива являются полноценными объектами класса Option. Они создаются по мере загрузки страницы браузером. Количество объектов Option, содержащихся в объекте document.f.s класса Select, можно узнать с помощью стандартного свойства массива: document.f.s.options.length. Кроме того, у самого объекта Select есть такое же свойство: document.f.s.length — оно полностью идентично предыдущему.

Программист имеет возможность не только создавать новые объекты Option, но и удалять уже созданные браузером объекты:

```
<FORM>
<SELECT NAME=s>
<OPTION>Первый вариант</OPTION>
<OPTION>Второй вариант</OPTION>
<OPTION>Третий вариант</OPTION>
</SELECT>
<INPUT TYPE=button VALUE="Удалить последний вариант"
    onClick="form.s.options[form.s.length-1]=null;">
<INPUT TYPE=reset VALUE="Сбросить">
</FORM>
```

В данном примере при загрузке страницы с сервера у нас имеются три альтернативы. Их можно просматривать как ниспадающий список вариантов. После нажатия на кнопку "Удалить последний вариант" в форме остается только две альтернативы. Если еще раз нажать на эту

Библиотека БГУИР

кнопку, останется только одна альтернатива. В конечном счете, вариантов не останется вовсе, т.е. пользователь лишится возможности выбора. При нажатии кнопки сброса (reset) варианты не восстанавливаются — альтернативы утеряны бесследно.

Теперь, используя конструктор Option, сделаем процесс обратимым:

```
<SCRIPT>
function RestoreOptions(){
   document.f.s.options[0] = new Option('Вариант один','',true,true);
   document.f.s.options[1] = new Option('Вариант два');
   document.f.s.options[2] = new Option('Вариант три');
   return false;
}
</SCRIPT>
<FORM NAME=f onReset="RestoreOptions();">
<SELECT NAME=s>
<OPTION>Первый вариант</OPTION>
<OPTION>Второй вариант</OPTION>
<OPTION>Третий вариант</OPTION>
</SELECT>
<INPUT TYPE=button VALUE="Удалить последний вариант"
     onClick="form.s.options[form.s.length-1]=null;">
<INPUT TYPE=reset VALUE=Reset>
</FORM>
```

Свойства selected и selectedIndex

Свойство **selectedIndex** объекта Select возвращает номер выбранного варианта (нумерация начинается с нуля).

```
<FORM> Вариант:
<SELECT onChange="form.e.value=selectedIndex;">
<OPTION>Один</OPTION>
<OPTION>Два</OPTION>
</SELECT>
Выбрали индекс: <INPUT NAME=e>
</FORM>
```

Обратите внимание, что в обработчике события onChange мы ссылаемся на второй элемент формы. На данный момент он не

Библиотека БГУИР

определен, но событие произойдет только тогда, когда мы будем выбирать вариант — к этому моменту поле уже будет определено.

Если список вариантов задан как <SELECT MULTIPLE>, т.е. с возможностью выбора нескольких опций одновременно, то свойство selectedIndex возвратит индекс первой выбранной опции. На этот случай имеется альтернатива: свойство **selected** у каждого объекта Option. Оно равно true, если данная опция выбрана, и false в противном случае. Пример будет приведен ниже.

Обработчик события onChange объекта Select

Событие Change наступает в тот момент, когда пользователь меняет свой выбор вариантов. Если поле является полем выбора единственного варианта, то все просто — см. предыдущий пример. Посмотрим, что происходит, когда мы имеем дело с полем выбора множественных вариантов:

```
<FORM>
Фрукты: <SELECT MULTIPLE
onChange="form.e.value='';
        for(i=0; i<length; i++)
        if(options[i].selected)
        form.e.value += options[i].text+', ';">

<OPTION>яблоко</OPTION>
<OPTION>банан</OPTION>
<OPTION>киви</OPTION>
<OPTION>персик</OPTION>
</SELECT><BR>
Выбраны позиции: <INPUT READONLY SIZE=70 NAME=e>
</FORM>
```

Обратите внимание на то, что событие Change происходит тогда, когда пользователь выбирает или отменяет какой-либо вариант. Исключение составляет тот случай, когда варианты при выборе последовательно отмечаются (нажатие кнопки мыши на одном элементе, ведение мыши до конечного элемента, отпускание кнопки мыши).

8.4. КНОПКИ

В HTML-формах используется четыре вида кнопок:

Библиотека БГУИР

```
<FORM>
<INPUT TYPE=button VALUE="Кнопка типа button">
<INPUT TYPE=submit VALUE="Кнопка отправки">
<INPUT TYPE=reset  VALUE="Кнопка сброса">
<INPUT TYPE=image SRC=a.gif> <!-- графическая кнопка -->
</FORM>
```

В атрибуте кнопки можно задать обработчик события onClick, а в атрибуте формы — обработчики событий onSubmit и onReset.

- при вызове метода click() кнопки **вызывается** и обработчик события onClick этой кнопки;
- при вызове метода submit() формы **не вызывается** обработчик события onSubmit формы;
- при вызове метода reset() формы **вызывается** и обработчик события onReset формы.

Кнопка button

Кнопка типа button вводится в форму главным образом для того, чтобы можно было выполнить какие-либо действия либо при ее нажатии пользователем, либо при вызове метода click().

Кнопка submit

Кнопка отправки (submit) позволяет отправить данные, введенные в форму, на сервер. В простейшем случае — при отсутствии у контейнера <FORM> атрибутов ACTION (его значением по умолчанию является адрес текущей страницы), METHOD (его значением по умолчанию является GET) и TARGET (его значением по умолчанию является _self) — стандартным действием браузера при отправке данных на сервер является просто перезагрузка текущей страницы

Убедитесь, что нажатие кнопки отправки приводит к следующей последовательности действий браузера:

- вызов обработчика события onClick у данной кнопки;
- вызов обработчика события onSubmit у формы;
- отправка данных формы на сервер.

Вызов метода **submit()** формы не равносилен нажатию кнопки отправки. При вызове этого метода будет выполнено только третье из

Библиотека ГГТУ им. П.О. Сухого

вышеперечисленных трех действий — отправка данных на сервер. То, что он не должен порождать вызов обработчика onClick кнопки отправки, вполне понятно — ведь мы пытаемся отправить данные в обход кнопки отправки (которой, кстати, может и не быть вовсе). Но и обработчик события onSubmit у формы тоже не вызывается. Тем самым данные могут уйти на сервер без предварительной проверки JavaScript-скриптом. Каким же образом заставить браузер вызвать обработчик onSubmit? Для этого существует возможность обратиться к этому обработчику напрямую: document.f.onsubmit(). Остается предусмотреть, что после этого метод submit() должен вызываться не всегда, а только если onSubmit либо не возвратил никакого значения, либо возвратил true, иными словами, если он не возвратил false. Окончательно мы получаем:

```
<FORM NAME=f ACTION="receive.htm"
     onSubmit="return confirm('Вы хотите отправить данные?')">
<INPUT onClick="alert('Вызван обработчик onClick у кнопки
отправки')"
      TYPE=submit VALUE="Кнопка отправки" NAME=s></FORM>

<A HREF="javascript:
   if(document.f.onsubmit() != false)
     document.f.submit();  void(0);">
   Вызвать <B>submit()</B> с преварительной проверкой
onSubmit</A>
```

Кнопка reset

Кнопка сброса (reset) позволяет вернуть все поля формы в первоначальное состояние, которое они имели при загрузке страницы. Нажатие кнопки сброса приводит к следующей последовательности действий браузера:

- вызов обработчика события onClick у данной кнопки;
- вызов обработчика события onReset у формы;
- восстановление значений по умолчанию во всех полях формы.

Графическая кнопка

Графическая кнопка — это разновидность кнопки отправки. Ее отличие в том, что вместо кнопки с надписью пользователь увидит картинку, по которой можно кликнуть:

```
<FORM ACTION="receive.htm">
```


Библиотека БГУИР

```
<INPUT TYPE=image SRC="pic.gif">
</FORM>
```

Кроме того, когда пользователь кликает по графической кнопке, то на сервер отправятся не только данные, введенные в поля формы, но также и координаты указателя мыши относительно левого верхнего угла изображения. К сожалению, перехватить эти координаты в JavaScript-программе не удается. Если Вам необходимо работать с этими координатами, то вместо графической кнопки рекомендуется создать активную карту с помощью контейнера <MAP>.

Графические кнопки имеют ряд странностей. Например, являясь одновременно и кнопкой, и изображением, они почему-то отсутствуют как в коллекции document.f.elements[], так и в коллекции document.images[] (IE 7, Mozilla Firefox). Как же обратиться к такой кнопке? Это можно сделать, например, задав атрибут ID:

```
<INPUT TYPE=image SRC="pic.gif" ID="d1">
```

и затем в программе написав: var **knopka = document.getElementById('d1')**. После этого мы можем обращаться к свойствам этой кнопки, например knopka.src, а также к методу knopka.click().

РАЗДЕЛ 9. ПРОГРАММИРОВАНИЕ ССЫЛОК

Для начала нам нужно разделить несколько понятий: применимость URL в атрибутах HTML-контейнеров; коллекция гипертекстовых ссылок; объекты класса URL.

- Адреса URL могут использоваться в атрибутах многих HTML-контейнеров, например:
- ссылки (URL в атрибуте HREF контейнера <A>);
- активные области (URL в атрибуте HREF контейнера <AREA>);
- картинки (URL в атрибуте SRC контейнера <IMG>);
- формы (URL в атрибуте ACTION контейнера <FORM>);
- внешние скрипты (URL в атрибуте SRC контейнера <SCRIPT>);
- связанные документы (URL в атрибуте HREF контейнера <LINK>).

Библиотека БГТУ им. П.О. Сухого

9.1. ОБЪЕКТЫ URL

Объект класса URL обладает свойствами, которые определены схемой URL. В качестве примера рассмотрим ссылку:

http://www.site.ru:80/dir/page.cgi?product=phone&id=3#mark

Тогда ее свойства примут следующие значения (обратите внимание, что значение поля search начинается со знака "?", а значение hash — со знака "#")

Свойство	Значение
href	http://www.site.ru:80/dir/page.cgi?product=phone&id=3#mark
protocol	http:
hostname	www.site.ru
port	80
host	www.site.ru:80
pathname	dir/page.cgi
search	?product=phone&id=3
hash	#mark

Свойство href является свойством по умолчанию. Это значит, что вместо window.location.href="..." можно писать window.location="...", а опуская window (который является объектом по умолчанию), можно писать location.href="..." и даже location="..." — эффект будет тот же: загрузится страница с новым адресом.

Коллекция ссылок links[]

К встроенным гипертекстовым ссылкам относятся собственно ссылки (**<A HREF=...>...</A>**) и ссылки "чувствительных" графических картинок. Все вместе они составляют коллекцию (встроенный массив) гипертекстовых ссылок документа document.links[].

К сожалению, обратиться по имени к гипертекстовой ссылке (т.е. как document.имя_ссылки) **нельзя**, даже несмотря на то, что у ссылки может быть задан атрибут NAME. Говоря точнее, такое обращение не рекомендуется в силу различий между браузерами. Поэтому обращаться к ним можно только через коллекцию ссылок по индексу: document.links[3] — это 4-я ссылка в документе. Стандарт также предусматривает обращение к ссылкам через коллекцию по имени:

Библиотека ГГТУ им. П.О.Сухого

document.links["имя_ссылки"], однако это работает не во всех браузерах (в Mozilla Firefox работает, в IE7 нет).

Две новые ссылки — это ссылки из контейнера MAP, который не отображается, но ссылки из него попадают в коллекцию ссылок links[]. При этом они могут попасть между обычными гипертекстовыми ссылками, если контейнер MAP расположить внутри текста документа. Итак, ссылки, создаваемые контейнерами <A HREF="..."> и <AREA HREF="...">, равноправно присутствуют в коллекции document.links[].

Замена атрибута HREF

Изменение части URL

Гипертекстовая ссылка — это объект класса URL, у которого помимо свойства href есть и другие, соответствующие частям URL, и их тоже можно менять.

9.1. СОБЫТИЯ URL

События MouseOver и MouseOut

Эти два события позволяют совершать действия при наведении или уводе курсора мыши на объекты, например, обесцвечивать и проявлять картинки, менять содержимое поля status и т.п. Применительно к гипертекстовым ссылкам, первое событие генерируется браузером, если курсор мыши находится над ссылкой, а второе — когда он покидает ссылку.

При наведении указателя мыши на ссылку мы будем подменять картинку, а при уводе указателя мыши с картинки — восстанавливать ее. В исходном HTML-документе это будет выглядеть следующим образом:

```
<A HREF="javascript:void(0);"
 onMouseOver="document.pic1.src='pic1.gif';"
 onMouseOut="document.pic1.src='pic1_.gif';">
 <IMG NAME=pic1 src=pic1_.gif BORDER=0></A>
```

Как уже рассказывалось в лекции 4, при возникновении события MouseOver у гиперссылки браузер показывает URL этой ссылки в поле статуса, а при возникновении события MouseOut восстанавливает в поле статуса прежнюю надпись. Перехватить первое событие (например, отменить вывод URL в строке статуса)

Библиотека БГТУ им. П.О. Сухого

можно, указав в его обработчике return true. Перехватить второе событие невозможно.

URL-схема "JavaScript:"

Для программирования гипертекстовых переходов в спецификацию URL разработчики JavaScript ввели, по аналогии с URL-схемами http, ftp и т.п., отдельную URL-схему javascript. Она уже упоминалась во вводной лекции при обсуждении вопроса о том, куда можно поместить JavaScript-программу в HTML-документе. Здесь мы рассмотрим ее подробнее.

Схема URL javascript: используется следующим образом:

<A HREF="JavaScript:код_программы">...</A>

<FORM ACTION="JavaScript:код_программы" ...> ... </FORM>

Рассмотрим пример гипертекстовой ссылки, в URL которой использована схема javascript:

<A HREF="javascript:alert('Спасибо!');">Кликните</A>

В общем случае, после "javascript:" может стоять произвольная программа JavaScript. Что в этом случае будет происходить, если кликнуть такую ссылку? Ответ следующий: сначала JavaScript-программа будет исполнена, в результате чего будет вычислено ее значение (которым всегда считается значение последнего оператора в программе). Затем, если это значение неопределено (undefined), то далее ничего не произойдет; если же полученное значение определено, то на экран будет выведена HTML-страница с этим результатом (преобразованным в строку и воспринятым как HTML-документ, который в том числе может содержать и HTML-тэги).

Если нужно, чтобы при клике по ссылке просто выполнились какие-то действия и более ничего не происходило (в частности, не происходил переход к какой-либо другой странице), то в конце программы можно поместить оператор **void(0)** либо **void(выражение)**. Эта функция вычисляет переданное в нее выражение и не возвращает никакого значения. Пример:

<A HREF="javascript: document.bgColor='green'; void(0);">Кликните</A>

Библиотека ГГТУ им. П.О. Сухого

Если кликнуть по такой ссылке, то изменится цвет фона, и больше ничего не произойдет. Без void(0) нам бы открылась web-страница со словом green.

Все сказанное справедливо и для использования URL-схемы "javascript:" в атрибуте ACTION контейнера <FORM>, за одним нюансом: в этом случае необходимо, чтобы в качестве метода доступа был указан METHOD="POST". Дело в том, что при использовании метода GET при отправке данных формы (т.е. при возникновении события Submit) к адресу URL, указанному в атрибуте ACTION, добавляется строка "?имя=значение&имя=значение&...", составленная из имен и значений элементов формы. Конечно же, такое выражение не является корректной JavaScript-программой, и браузеры могут выдавать сообщения об ошибке (а могут и не выдавать, а просто не выполнять программу). Пример:

```
<FORM NAME=f METHOD=post
ACTION="javascript:alert('Длина строки = '
                    +document.f.e.value.length);">
<INPUT NAME=e>
<INPUT TYPE=submit VALUE="Длина">
</FORM>
```

Обработка события Click

У гипертекстовой ссылки помимо URL, указанного в атрибуте HREF, можно указать действия, которые браузер должен выполнить, когда пользователь кликнет по данной ссылке, перед тем, как перейти по указанному URL. Соответствующая программа JavaScript называется обработчиком события Click и помещается в атрибут onClick контейнера <A>. Если обработчик события возвращает значение false (это можно реализовать путем помещения в конец обработчика команды return false), то переход по адресу URL, указанному в атрибуте HREF, не будет совершен. Если же обработчик возвращает true либо ничего не возвращает, то после выполнения обработчика события будет совершен переход по адресу URL. Например:

```
<A onClick="return confirm('Хотите посетить сайт INTUIT?')"
 HREF="http://www.intuit.ru/">Перейти на сайт INTUIT</A>
```

В этом примере confirm() возвращает либо true, либо false, в зависимости от того, на какую кнопку нажмет пользователь в

Библиотека ГГТУ им. П.О. Сухого

предложенном запросе. Соответственно, переход на указанный адрес либо произойдет, либо нет. Если бы мы в этом примере опустили слово return, то обработчик события ничего бы не возвращал (независимо от действий пользователя на запрос confirm) и переход на указанный URL совершался бы в любом случае.

И последнее замечание. Часто для того, чтобы скрипт запускался, когда посетитель кликает ссылку, программисты пишут что-то такое:

<A HREF="#" onClick="программа JavaScript">...</A>

<A HREF="javascript:void(0)" onClick="программа JavaScript">...</A>

<A HREF="javascript: программа JavaScript">...</A>

При этом код выполняется и, на первый взгляд, все нормально. Но на второй взгляд становится видно, что после клика на ссылку могут прекратить грузиться недогруженные элементы страницы (большие картинки и т.п.), останавливаются анимированные GIF'ы и, быть может, происходит что-то еще из этой серии. Решение здесь такое: если вы рассчитываете при клике пользователя по ссылке оставить его на текущей странице, то не забудьте прописать выход return false из обработчика события onClick, например:

<a href="#" onClick="программа JavaScript; return false;">

В этом случае после вызова программы JavaScript выполнение клика по ссылке прекратится, поэтому браузер не будет считать, что произошел переход на другую страницу.

РАЗДЕЛ 10. COOKIES. НЕВИДИМЫЙ КОД И БЕЗОПАСНОСТЬ

10.1. ПОНЯТИЕ COOKIE

Формально, cookie (читается: куки; не склоняется) — это небольшой фрагмент данных, которые веб-браузер пересылает веб-серверу в HTTP-запросе при каждой попытке открыть очередную страницу сайта. Обычно куки создаются веб-сервером и присылаются в браузер при первом запросе к сайту. Куки также могут быть созданы самой загруженной web-страницей, а именно имеющимся в ней скриптом JavaScript. Далее хранятся на компьютере пользователя

Библиотека БГУИР ГГТУ им. П.О.Сухого

в виде текстового файла, до тех пор, пока либо не закончится их срок, либо они будут удалены скриптом или пользователем.

На практике cookie обычно используются для:

- аутентификации пользователя;
- хранения персональных предпочтений и настроек пользователя;
- отслеживания состояния сессии доступа пользователя;
- ведения статистики о пользователях.

Без куки просмотр каждой веб-страницы является изолированным действием, не связанным с просмотром других страниц того же сайта. В конце этого раздела мы создадим страницы, передающие друг другу информацию с помощью куки.

Главными атрибутами cookie являются: имя/значение куки, срок действия, путь, доменное имя, шифрование. Атрибуты записываются через точку с запятой. Обязательным является лишь имя/значение. Например:

customer=21584563; expires=Fri, 31-Dec-2010 23:59:59 GMT; path=/; domain=www.shop.ru; secure

Здесь customer — имя куки; 21584563 — значение куки; expires=... — срок действия куки; path=/ — путь, для которого действует эта куки (в данном случае куки действуют для любой страницы на данном сайте); domain=www.shop.ru — домен (сайт), для которого будет действовать эта куки; secure — использовать ли для передачи куки шифрованный канал (HTTPS). Если не указан expires, то куки действительна до закрытия браузера. Если не указаны path и domain, то куки действительна для текущей web-страницы. Удаляются куки путем указания истекшего срока действия (браузер сам сотрет такие куки по окончании своей работы). Подробнее о формате cookie рассказано в лекции Спецификация Cookie из соседнего курса тех же авторов. Здесь же мы остановимся на методах работы с cookie с помощью инструментов JavaScript.

Чтение cookie

Для работы с куки из сценария JavaScript используется свойство document.cookie. Следующая команда покажет все установленные куки: **alert(document.cookie);**

Она выдаст нечто вроде: **name1=value1; name2=value2; ...,** т.е. перечисление пар имя=значение, разделенных точкой с запятой и

Библиотека БГТУ им. П.О. Сухого

пробелом. Как видите, нам показывают только имена и значения куки; другие атрибуты куки (срок действия, домен и т.д.) через свойство document.cookie недоступны.

Но обычно требуется больше — узнать, установлено ли значение конкретной куки, и если установлено, то прочитать его. Значит, нужно разобрать полученную выше строку с помощью методов работы со строковыми объектами. Для этого создадим две функции: existsCookie — проверяет, имеется ли куки с данным именем; CookieValue — возвращает значение куки по ее имени:

```
<SCRIPT>
function existsCookie(CookieName){
  // Узнает, имеется ли куки с данным именем
  return (document.cookie.split(CookieName+'=').length>1);
}

function cookieValue(CookieName){
  // Выдает значение куки с данным именем
  var razrez = document.cookie.split(CookieName+'=');
  if(razrez.length>1) {
    // Значит, куки с этим именем существует
    var hvost = razrez[1],tzpt = hvost.indexOf(';'),
    EndOfValue = (tzpt>-1)? tzpt : hvost.length;
    return unescape(hvost.substring(0,EndOfValue));
  }
}
</SCRIPT>
```

Создание или изменение cookie

Далее мы хотим создавать или менять cookie. Разработчики языка JavaScript позаботились о web-программистах и реализовали свойство document.cookie довольно интеллектуально. Если выполнить простую команду присвоения:

document.cookie='ИмяКуки=Значение; expires=дата; path=путь; domain=домен; secure';

то прежние хранившиеся куки не будут стерты, как можно заподозрить. Вместо этого браузер проверит, не имеется ли уже в document.cookie куки с именем ИмяКуки. Если нет, то новая куки будет добавлена в document.cookie; если да, то для куки с этим

Библиотека БГУИР

именем будут обновлены указанные в команде параметры (значение, срок действия и т.д.).

Напишем универсальную функцию для задания куки, которой Вы можете пользоваться на практике. Первые два ее аргумента (name и value) обязательны, остальные необязательны.

```
<SCRIPT>
function setCookie(name, value, exp, pth, dmn, sec)
{ // Создает cookie с указанными параметрами
 document.cookie = name + '=' + escape(value)
 + ((exp)? '; expires=' + exp : '')
 + ((pth)? '; path='    + pth : '')
 + ((dmn)? '; domain='  + dmn : '')
 + ((sec)? '; secure'        : ''); }
</SCRIPT>
```

На практике заранее известно не время истечения срока действия куки, а сам срок действия (в днях, часах, минутах и т.д.), отсчитывая от текущего момента. Напишем функцию, которая по этим данным возвращает точный момент времени, причем в нужном нам формате:

```
<SCRIPT>
function TimeAfter(d,h,m){
   // Выдает время через d дней h часов m минут
   var now = new Date(),    // объект класса Data
   nowMS = now.getTime(), // в миллисекундах (мс)
   newMS = ((d*24 + h)*60 + m)*60*1000 + nowMS;
   now.setTime(newMS);    // новое время в мс
   return now.toGMTString();
}
</SCRIPT>
```

Удаление cookie

Удалить куки — значит в качестве времени истечения куки указать какой-либо прошлый момент времени, например, "сутки назад". Напишем соответствующую функцию:

```
function deleteCookie(CookieName){
   // Удаляет куки с данным именем
   setCookie(CookieName,'',TimeAfter(-1,0,0));
}
```

Демонстрационный пример

Теперь у нас есть весь арсенал функций работы с куки; сохраните их единый в файл cookies.js. Мы создадим две независимые web-страницы, которые благодаря куки смогут обмениваться информацией. Первая страница запрашивает имя пользователя и записывает его в куки сроком на 1 минуту.

```
<SCRIPT SRC='cookies.js'></SCRIPT>
<SCRIPT>
   setCookie('customername', prompt('Введите ваше имя',''),
TimeAfter(0,0,1)  );
   alert('Мы Вас запомнили!');
</SCRIPT>
```

Вторая страница приветствует пользователя по имени, которое она прочитает из cookie. Убедитесь, что она узнает пользователя, даже если перед открытием страницы закрыть браузер. Однако если открыть эту страницу через 1 минуту, то она уже не сможет узнать пользователя.

```
<SCRIPT SRC='cookies.js'></SCRIPT>
<SCRIPT>
   if(existsCookie('customername'))
       alert('Приветствуем Вас, ' +CookieValue('customername')+ '!');
   else alert('Извините, мы Вас уже не помним...')
</SCRIPT>
```

10.2. НЕВИДИМЫЙ КОД

Вопрос доступности JavaScript-кода рассматривается с двух точек зрения: идентификация, как следствие — необходимость сокрытия кода, и безопасность пользователя, следовательно — доступность кода.

Приемы программирования со скрытым кодом позволяют решить еще несколько задач, которые не связаны с безопасностью.

Мы будем рассматривать возможности использования скрытого кода, не вынося вердиктов о преимуществе того или иного подхода. Рассмотрим несколько вариантов:

- невидимый фрейм;
- код во внешнем файле;
- обмен данными посредством встроенной графики.

Библиотека ГГТУ им. П.О.Сухого

Строго говоря, первые два варианта не скрывают код полностью. Они рассчитаны либо на неопытных пользователей, либо на нелюбопытных. Так или иначе, не каждый же раз вы будете смотреть исходный текст страницы.

Невидимый фрейм

Технология программирования в невидимом фрейме основана на том, что при описании фреймовой структуры можно задать конфигурацию типа:

```
<FRAMESET COLS="100%,*" BORDER=0>
  <FRAME NORESIZE SRC=left.htm>
  <FRAME NORESIZE SRC=right.htm>
</FRAMESET>
```

В этом случае левый фрейм займет весь объем рабочей области окна, а содержание правого будет скрыто. Именно в этом невидимом фрейме мы и разместим код программы (например, приведенный выше скрипт считывания полей из формы в левом фрейме). В невидимый фрейм иногда помещают функции подкачки графики, позволяя пользователю уже работать с основным фреймом, пока грузится остальная часть графики.

Код во внешнем файле

О том, как подключать код JavaScript, размещенный во внешнем файле, рассказывалось во вводной лекции:

```
<SCRIPT SRC="myscript.js"></SCRIPT>
```

Данный способ позволяет скрыть код лишь от ленивого пользователя. Но сам код JavaScript легко доступен, т.к. указанный файл можно просто скачать отдельно, либо сохранить всю HTML-страницу (со всеми подключенными к ней скриптами) с помощью меню браузера.

Обмен данными посредством встроенной графики

Данный прием основан на двух идеях: возможности подкачки графического образа без перезагрузки страницы и возможности подкачки этого графического образа не через указание URL графического файла, а через CGI-скрипт, который возвращает Content-type: image/... или осуществляет перенаправление. При этом следует учитывать, что использовать метод, отличный от GET, можно

только в формах. В следующем примере мы создали функцию change_image(), которая формально говоря меняет значение свойства src картинки. Но в качестве побочного эффекта позволяет серверу узнать, установлены ли у пользователя cookie (если соответствующим образом запрограммировать CGI-скрипт image.cgi на стороне сервера):

```
<SCRIPT>
function change_image(x) {
   document.x.src = 'http://abc.ru/image.cgi?' + document.cookie;
}
</SCRIPT>
<A HREF="javascript:change_image(i);"><IMG NAME=i
SRC=image1.gif></A>
```

Эта безобидная последовательность операторов JavaScript позволит нам узнать получил ли клиент cookie. Куки могут не поддерживаться по разным причинам. В данном случае программа передает на сервер выставленные им cookie в качестве параметра скрипта под видом изменения картинки.

10.3. МОДЕЛЬ БЕЗОПАСНОСТИ

При программировании на JavaScript потенциально существует возможность доступа из программы к персональной информации пользователя. Такая проблема возникает всегда, когда нечто, запускаемое на компьютере, имеет возможность самостоятельно организовать обмен данными по сети с удаленным сервером. От версии к версии управление защитой таких данных постоянно совершенствуется.

По умолчанию к защищенным в JavaScript данным относятся:

Объект или класс	Свойства
document	cookie, domain, lastModified, location, referrer, title, URL, links[], forms[]
form	action
элемент формы	checked, defaultChecked, name, value, defaultValue, selectedIndex, toString()
history	current, next, previous, toString(), все элементы массива посещенных адресов
Location, Link, Area	hash, host, hostname, href, pathname, port,

Библиотека ГГТУ им. П.О. Сухого

	protocol, search, toString()
Option	defaultSelected, selected, text, value
Window	defaultStatus, status

Защищенными эти данные являются с той точки зрения, что программа не может получить значения соответствующих атрибутов. Главным образом речь здесь идет о программе, которая пытается получить доступ к данным, которые определены на другой странице (не на той, в рамках которой данная программа исполняется). Например, к данным из другого окна.

В настоящее время известны три модели защиты: **запрет на доступ** (Navigator 2.0), **taint model** (Navigator 3.0), **защита через Java** (Navigator 4.0). Применение моделей и соответствующие приемы программирования — это отдельный сложный вопрос, требующий знаний и навыков программирования на языке Java, поэтому в рамках данного курса мы его рассматривать не будем.

Отметим только, что к большинству свойств объектов текущего окна программист имеет доступ. Они становятся защищенными только в том случае, если относятся к документу в другом окне и загруженному из другого Web-узла. Поэтому ограничения, накладываемые системой безопасности JavaScript, достаточно гибкие и не очень сильно мешают разработке страниц с применением этого языка программирования.

РАЗДЕЛ 11.СОБЫТИЯ КЛАВИАТУРЫ И МЫШИ

11.1. СОБЫТИЯ

В объекте события содержится подробнейшая информация о том, что и где произошло.

К сожалению, здесь много кроссбраузерных несовместимостей, однако самые важные из них легко преодолимы.

Тип события

Тип события можно получить, используя кроссбраузерное свойство type объекта событие.

```
function getEventType(e) {
if (!e) e = window.event;
alert(e.type);
```


Библиотека ГГТУ им. П.О.Сухого

```
}
```

Данный код в действии (в примере обрабатываются события click и mouseout):

Элементы, связанные с событием

Элемент-триггер: target

Чаще всего нужно узнать, на каком элементе сработало событие.

Например, мы поймали на внешнем div'е и хотим знать, на каком из внутренних элементов оно на самом деле произошло.

В Internet Explorer у объекта window.event для этого есть свойство srcElement, в остальных браузерах, работающих по рекомендациям W3C, для этого используется event.target.

Вот пример использования этого свойства. Обработчик стоит только на внешнем диве, но благодаря event.target выводит по клику класс исходного элемента.

```
<div class="d1"
onclick="t=event.target||event.srcElement; alert(t.className)">
<span class="number">1</span>
<div class="d2">
<span class="number">2</span>
<div class="d3">
<span class="number">3</span>
</div>
<a class="d2a" href="javascript:void(0)">ссылка</a>
</div>
</div>
```

Javascript-обработчик в примере висит только на внешнем диве d1 и выглядит примерно так:

```
function(event) {
   // получить объект событие.
   // вместо event лучше писать window.event
   event = event || window.event
   // кросс-браузерно получить target
   var t = event.target || event.srcElement
   alert(t.className)
}
```

Когда триггеров больше одного: relatedTarget

Для событий mouseout и mouseover предусмотрен способ получить как элемент на который курсор мыши перешел, так и элемент, с которого он перешел.

Эти свойства - relatedTarget в W3C, и fromElement/toElement в Internet Explorer.

```
// Обработчик для mouseover
function mouseoverHandler(event) {
    event = event || window.event
    var relatedTarget = event.relatedTarget || event.fromElement
    // для mouseover
    // relatedTarget - элемент, с которого пришел курсор мыши
}

// Обработчик для mouseout
function mouseoutHandler(event) {
    event = event || window.event
    var relTarg = event.relatedTarget || event.toElement
    // для mouseout
    // relatedTarget - элемент, на который перешел курсор мыши
}
```

Свойство relatedTarget дополняет target. В нем всегда находится информация о втором элементе, участвовавшем в событии.

Поэтому его можно получить для IE, взяв то свойство из fromElement/srcElement, которое не равно target:

```
if (!e.relatedTarget && e.fromElement) {
    e.relatedTarget = (e.fromElement==e.target) ? e.toElement :
e.fromElement
}
```

Текущий элемент: this

При всплытии - событие по очереди взывает обработчики на элементе-триггере и дальше, вверх по документу.

По мере всплытия, *текущим* элементом каждый раз становится новый. Иначе говоря. *текущий* элемент - это тот, к которому в данный момент "доплыло" событие.

Стандартный способ получить текущий элемент - использовать переменную this.

```
<div class="d1" onclick="highlightMe(this)">1
```

```
<div class="d2" onclick="highlightMe(this)">2
<div class="d3" onclick="highlightMe(this)">3</div>
</div>
</div>
```

Показ ошибок JavaScript

Самый прсотой способ узнать, какая ошибка произошла – вывести сообщение об этом. Для этого в скрипте, который должен быть расположен ближе к верхней части документа, создадим обработчик соответствующего события onerror, которое имеется в объекте window.

```
window.onerror = function(msg, src, line) {
  alert("Ошибка: "+msg+"\n"+"Файл: "+src+"\n"+"Строка: "+line);
};
```

Теперь при возникновении ошибки будет выведено сообщение, в котором будет виден тип ошибки, файл, в котором она возникла и даже номер строки с ошибкой.
Заметим, что подобная конструкция не будет выдавать корректное сообщение в браузере IE по описанным выше причинам.

11.2. СОБЫТИЯ МЫШИ

Самое известное событие - onclick. Более полный список:

onmousedown	Нажатие на кнопку мыши
onmouseup	Нажатая кнопка мыши отпущена
onclick	Клик мыши
ondblclick	Двойной клик
oncontextmenu	Правый клик

События mousedown и mouseup в основном используются, когда идет нажатие на кнопку, перемещение, а потом мышь отпускается. Например, при выделении текста, или переносе объекта.

А события click и dblclick в основном нужны для работы с простыми кликами. Клик происходит при последовательных mousedown->mouseup на одном и том же объекте. Если Вы передвините мышь куда-то между mousedown и mouseup, то событие click не произойдет.

Библиотека ГГТУ им. П.О.Сухого

Событие contextmenu возникает при правом клике мышью, и по умолчанию вызывает контекстное меню. Не на всех браузер показ меню можно отключить.

mouseover, mouseout и mousemove

События **mouseover** и **mouseout** срабатывают каждый раз, когда мышь заходит на элемент или выходит с него.

Событие **mousemove** срабатывает при каждом передвижении мыши. Если Вы пишите код для обработки mousemove - постарайтесь сделать его достаточно быстрым, иначе при передвижении курсора будут заметны тормоза.

Порядок и частота

При переходе с внешнего элемента на внутренний, фиксируется событие onmouseout для внешнего и onmouseover для внутреннего. Смысл в том, что мышь находится ровно "в одном элементе" одновременно: в ближайшем(по z-index) и самом глубоком.

Эти события обладают еще одним забавным свойством. А именно - частота их регистрации соответствует скорости движения мыши. Как браузер успеет зарегистрировать событие - так и получится.

Можно зайти во внутренний элемент, не проходя через внешний.

Иначе говоря, если мышь движется быстро, то события для внешнего элемента могут оказаться пропущенными.

Точно также можно выйти из внутреннего элемента, не проходя через внешний. Всего-то надо быстро двигать мышкой.

Кнопка мыши: which/button.

После получения события обычно интересно узнать, какая кнопка была нажата. Если, конечно, это не событие contextmenu, с которым все и так понятно.

Для этого в объекте event есть два свойства: event.which и event.button, которые содержат числовые значения, соответствующие нажатой кнопке. К сожалению, тут есть некоторые несовместимости.

		Internet Explorer	**Firefox, Safari Win, Opera**	**Konqueror**
ЛЕВАЯ	event.which	undefined	1	1

Библиотека ГГТУ им.П.О.Сухого

КНОПКА	event.button	1	0	1
СРЕДНЯЯ КНОПКА	event.which	undefined	2	2
	event.button	4	1	4
ПРАВАЯ КНОПКА	event.which	undefined	3	3
	event.button	2	2	2

Свойство event.which было изначально изобретено Netscape, а event.button использовалось в Internet Explorer.

Через некоторое время браузеры стали использовать оба и все перепуталось.

Сравнение подходов

Стандарт/Firefox

В стандарте W3C прописано свойство button, которое ведет себя, как в Firefox, т.е:

- - левая кнопка
- - средняя кнопка
- - правая кнопка

IE

Подход создателей браузера Internet Explorer, вообще говоря, более универсальный, так как button является 3-битовым числом, каждый бит которого отвечает за кнопку мыши.

Так, button & 1 (первый бит) установлен в 1, если нажата левая кнопка, button & 2 (второй бит) установлен в 1, если нажата правая кнопка, и button & 4 (третий бит) - если нажата средняя.

В результате мы не можем отловить, когда, например, нажаты левая и правая кнопки, а когда только левая или только правая. К сожалению, это можно сделать только в IE.

Кросс-браузерный код

Удобнее всего - взять за основу свойство which, которое одинаково поддерживают почти все браузеры.

Остается лишь составить which из button для Internet Explorer:

```
if (!e.which && e.button) {
  if (e.button & 1) e.which = 1
  else if (e.button & 4) e.which = 2
  else if (e.button & 2) e.which = 3
}
```


Библиотека ГГТУ им.П.О.Сухого

Координаты мыши: clientX(Y)/pageX(Y)

При обработке событий, связанных с мышью, нужен кроссбраузерный способ получения координат курсора из события в обработчике.

Относительно окна

Координаты курсора мыши относительно окна находятся в стандартных свойствах **clientX/clientY**. Они одинаково поддерживается всеми браузерами.

Если у вас есть окно 500x500, и мышь находится в центре, то clientX и clientY будут оба равны 250. Если вы затем проскроллируете документ вниз, налево или вверх, не двигая курсор - значения clientX/clientY не изменятся, так как отсчитываются относительно окна, а не документа.

Относительно документа

Как правило, при обработке события нужна позиция мыши относительно документа, учитывающая прокрутку. Стандарт W3C предоставляет для этого свойство **pageX/pageY**.

Если у вас есть окно 500x500, и мышь находится в центре, то pageX и pageY будут оба равны 250. Если вы затем проскроллируете на 250 пикселей вниз, pageY станет равным 750.

Таким образом pageX/pageY содержат координаты, на каком месте документа произошло событие, учитывая все прокрутки.

Свойства pageX/pageY поддерживаются всеми браузерами, кроме Internet Explorer.

В IE их можно получить из clientX/clientY, прибавив к ним **scrollLeft/scrollTop**.

Обычно оно находится в <body>: document.body.scrollLeft, но это не всегда так. Например, при выборе Strict DTD оно высчитывается для <html>: document.documentElement.scrollLeft. Кроме того, тэга <body> может просто не быть в документе.

Поэтому мы сначала возьмем html.scrollLeft (если есть), затем проверим body.scrollLeft. Если нет ни того, ни того, то 0.

```
var html = document.documentElement
var body = document.body
e.pageX = e.clientX + (html && html.scrollLeft ||
                       body && body.scrollLeft || 0)
```

Кроме того, document в IE может быть немного сдвинут с позиции 0,0. Значение сдвига находится в document.documentElement.clientLeft/clientTop, и его также необходимо учесть.

Этот код позволяет надежно получить pageX/pageY для IE, в котором его изначально нет:

```
if (e.pageX == null && e.clientX != null ) {
   var html = document.documentElement
   var body = document.body
   e.pageX = e.clientX + (html && html.scrollLeft || body &&
               body.scrollLeft || 0) - (html.clientLeft || 0)
   e.pageY = e.clientY + (html && html.scrollTop || body &&
               body.scrollTop || 0) - (html.clientTop || 0)
}
```

11.3. СОБЫТИЯ КЛАВИАТУРЫ

Здесь мы рассмотрим основные «клавиатурные» события и работу с ними.

Их три:

keydown

Вызывается при нажатии любой клавиши, позволяет получить её *скан-код*.

Скан-код для клавиши одинаков в любой раскладке и в любом регистре. Например, клавиша «z» может означать символ «z», «Z» или «я», «Я» в русской раскладке, но её *скан-код* будет всегда одинаков: 90.

keypress

Возникает сразу после keydown, если нажата *символьная* клавиша, т.е. нажатие приводит к появлению символа.

Любые буквы, цифры генерируют keypress. Управляющие клавиши, такие как Ctrl, Shift, F1, F2.. — keypress не генерируют.

Это событие позволяет получить *код символа*, т.е. в отличие от keydown — отличить «z» от «я». Поэтому оно используется тогда, когда нам важно знать именно символ.

Библиотека ГГТУ им. П.О. Сухого

keyup

Клавиша отпущена, позволяет узнать скан-код.

Между событиями keypress и keydown есть существенное различие.

Конец формы

```
document.getElementById('kinput').onkeydown = khandle;
document.getElementById('kinput').onkeyup = khandle;
document.getElementById('kinput').onkeypress = khandle;
function khandle(e) {
    e = e || event;
    if (document.forms.keyform[e.type + 'Ignore'].checked) return;

    var evt = e.type;
    while (evt.length < 10) evt += ' '
    showmesg(evt +
    ' keyCode=' + e.keyCode +
    ' which=' + e.which +
    ' charCode=' + e.charCode +
    ' char=' + String.fromCharCode(e.keyCode || e.charCode) +
    (e.shiftKey ? ' +shift' : '') +
    (e.ctrlKey ? ' +ctrl' : '') +
    (e.altKey ? ' +alt' : '') +
    (e.metaKey ? ' +meta' : ''), 'key'
    )

    if (document.forms.keyform[e.type + 'Stop'].checked) {
        e.preventDefault ? e.preventDefault() : (e.returnValue = false);
    }
}
```

Неизвестные свойства и методы мы подробно рассмотрим ниже.

Попробуйте нажать *печатные клавиши*, например ‘S’, ‘1’ или ‘,’.

Нажатие на них вызывает сначала keydown, а затем keypress. Когда клавишу отпускают, срабатывает keyup.

Попробуйте нажать *специальные клавиши*, например Alt, Shift, Delete или Стрелка Вверх — будет keydown -> keyup, но не keypress.

Вызывается keydown при нажатии и keyup в момент отпускания.

Попробуйте нажать *Alt+печатная клавиша*, например Alt+E. Обратите внимание: Alt+клавиша в некоторых браузерах генерирует keypress, а в некоторых (IE, Chrome). Оба варианта имеют право на существование, т.к. такое сочетание не дает печатный символ.

Нажимая другие клавиши, вы заметите, что браузеры Firefox и Opera генерируют keypress для многих специальных клавиш, а IE инициирует keypress для Esc.

Это не совсем правильно, т.к. смысл события — узнать о том, что введён *символ*, а управляющая клавиша символа не даёт. Но это «лишние» события легко отфильтровать. Мы увидим это далее, при обсуждении того, как получить символ из keypress.

11.4. СВОЙСТВА КЛАВИАТУРНЫХ СОБЫТИЙ

keyCode

Это *скан-код* клавиши. Например, если нажали клавишу «a», то символом может быть «a» или «A» (или символ из другого языка), но keyCode остаётся тем же: 97. Он зависит только от клавиши, а не от символа.

charCode

Это *код символа* по кодировке ASCII.

which

Нестандартное свойство, гибрид charCode и keyCode, созданное с единственной целью — запутать разработчика.

Однако и оно бывает полезным, когда нужно получить символ в keypress. Мы увидим это чуть дальше, когда будем разбирать кросс-браузерный код.

shiftKey, ctrlKey, altKey, metaKey

Это булевы свойства и отображают только состояние соответствующей клавиши: Shift, Ctrl, Alt или Command(для Mac).

Получение символа из keypress

Основные браузерные особенности, важные для получения символа из события keypress — следующие:

1. Во всех браузерах, кроме IE, у события keypress есть свойство charCode, которое содержит код символа.

Библиотека ГГТУ им.П.О.Сухого

2. При этом у Opera есть некоторые баги со специальными клавишами. Для некоторых из них, она «забывает» указать charCode, например, для «backspace».

3. Браузер IE в случае с keypress не устанавливает charCode, а записывает код символа в keyCode (обычно там хранится скан-код).

Вы можете сильно не задумываться над ними, т.к. достаточно одной функции, которая их учитывает и возвращает символ.

Функция для получения символа из события keypress:

```
// event.type должен быть keypress
function getChar(event) {
  if (event.which == null) {
    return String.fromCharCode(event.keyCode) // IE
  } else if (event.which!=0 && event.charCode!=0) {
    return String.fromCharCode(event.which)  // остальные
  } else {
    return null; // специальная клавиша
  }
}
```

Обратите внимание на последний случай. Специальные клавиши не имеют соответствующих символов, поэтому попытка получения такого символа через String.fromCharCode дала бы странные результаты.

Вместо этого мы фильтруем их проверкой event.which!=0 && event.charCode!=0. Она гарантирует, что клавиша не является специальной, даже в старых браузерах и в Opera.

Неправильный getChar

В сети вы можете найти другую функцию того же назначения:

```
function getChar(event) {
  return String.fromCharCode(event.keyCode || event.charCode);
}
```

Она работает неверно для многих специальных клавиш, потому что не фильтрует их. Например, она возвращает символ амперсанда "&", когда нажата клавиша 'Стрелка Вверх'.

Отмена пользовательского ввода

Результатом нажатия неспециальной клавиши, как правило, является символ. Но появление символа можно предотвратить.

Библиотека БГТУ им. П.О. Сухого

Для отмены ввода можно использовать оба события: keydown и keypress.

Попробуйте что-нибудь ввести в этих полях:

<input onkeydown="return false" type="text" size="30">

<input onkeypress="return false" type="text" size="30">

Попробуйте что-нибудь ввести в этих полях (не получится):

В IE и Safari/Chrome отмена действия браузера при keydown также предотвращает само событие keypress. Попробуйте это на тестовом стенде: отмените keydown и наберите что-нибудь. В IE и Safari/Chrome keypress не появится.

Демо: перевод вводимых символов в верхний регистр

Это поле переводит все вводимые символы в верхний регистр:

```
<input id='my' type="text">
<script>
document.getElementById('my').onkeypress = function(event) {
  var char = getChar(event || window.event);
  if (!char) return;
      this.value = char.toUpperCase();
  return false;
}
</script>
```

Все символы, которые вы введете сюда, перейдут в верхний регистр:

В примере выше, действие браузера отменено с помощью return false, а вместо него в input добавлено наше собственное значение.

С этим виджетом есть проблема. Если навести курсор на середину поля и ввести что-нибудь, то введенное добавится в конец. Что делать?

Можно узнать, на какой позиции находится курсор, получив его из свойства input.selectionStart (для IE<9 понадобится дополнительный код). Еще один выход — работать со значением уже после того, как новый символ туда добавится, но тогда его увидит посетитель, и это будет не так красиво.

Впрочем, в 90% случаев событие keypress используют для получения и возможной отмены ввода, а не его изменения, поэтому на практике эта проблема возникает редко.

Нам нужно проверять *символы* при вводе, поэтому, будем использовать событие keypress.

Библиотека БГТУ им. П.О. Сухого

Алгоритм такой: получаем символ и проверяем, является ли он цифрой. Если не является, то отменяем действие по умолчанию.

Кроме того, игнорируем специальные символы и нажатия со включенным Ctrl/Alt.

Итак, вот решение:

```
input.onkeypress = function(e) {
  e = e || event;
  if (e.ctrlKey || e.altKey) return;
  var chr = getChar(e);
  if (chr == null) return;
  if (chr < '0' || chr > '9') {
    return false;
  }
}
```

Работа со скан-кодами: keydown/keyup

Иногда нам нужно узнать только клавишу, а не символ. Например, специальные клавиши типа стрелок, ввода, F1..F12 — у них нет символов.

Большинство браузеров не генерируют для них событие keypress. Вместо него нужно использовать keydown/keyup.

Хорошая новость заключается в том, что все современные браузеры и даже IE придерживаются одних и тех же кодов для специальных клавиш(за исключением брендовых клавиш, типа клавиши со значком Windows).

Актуальный пример — «горячие» клавиши. Когда мы делаем свою собственную «горячую» клавишу на JavaScript, она должна работать независимо от регистра и текущего языка раскладки. Нас не интересует символ. Нам нужен только скан-код. Поэтому её следует ловить на keydown/keyup.

Скан-коды VS коды символов

Для всех буквенных, циферных и большинства специальных клавиш, скан-код совпадает с кодом символа.

В случае букв/цифр, скан-код будет равен коду заглавной английской буквы/цифры.

Например, если вы хотите отследить нажатие «Ctrl-S». Проверяющий код для keydown будет таким:

e.ctrlKey && e.keyCode == 'S'.charCodeAt(0);

Библиотека ГГТУ им. П.О. Сухого

Для keyCode не имеет значения, нажал пользователь «s» или «S» или у него была русская раскладка.

Скан-код не равен коду символа для большинства пунктуационных знаков, включая скобки и арифметические знаки.

В этом случае его можно получить на тест-стенде.

Тут есть небольшая кросс-браузерная несовместимость. Например, у клавиши, обозначающей дефис "-", в IE keyCode=189, а в других браузерах keyCode=109. При этом её charCode везде равен 45.

Такое несовпадение кодов IE и других браузеров возникает лишь с клавишами ';', '=' и '-'. С остальными клавишами всё в порядке.

Специальные действия

Некоторые специальные действия можно отменить. Например, если была нажата клавиша backspace, но keydown вернул false, то символ не будет удален.

Конечно же, есть действия, которые в принципе нельзя отменить, в первую очередь — те, которые происходят на уровне операционной системы. Комбинация Alt+F4 закрывает браузер в большинстве операционных систем, что бы вы ни делали в JavaScript.

Создайте функцию runOnKeys(func, code1, code2, ... code_n), которая запускает func при одновременном нажатии клавиш со скан-кодами code1, code2, …, code_n.

Например, код ниже выведет alert при одновременном нажатии клавиш "Q" и "W" (в любом регистре, в любой раскладке)

```
runOnKeys(
 function() { alert("Привет!") },
 "Q".charCodeAt(0),
 "W".charCodeAt(0)
);
```

Функция runOnKeys — с переменным числом аргументов. Для их получения используйте arguments.

Используйте два обработчика: document.onkeydown и document.onkeyup. Первый отмечает нажатие клавиши в объекте pressed = {}, устанавливая pressed[keyCode] = true а второй — удаляет это свойство. Если все клавиши с кодами из arguments нажаты — запускайте func.

Возникнет проблема с повторным нажатием сочетания клавиш после alert, решите её.

Библиотека БГУИР

РАЗДЕЛ 12. РАБОТА С ЭЛЕМЕНТАМИ DOM

12.1. ПОИСК ЭЛЕМЕНТОВ В DOM

Стандарт DOM предусматривает несколько средств поиска элемента. Это методы getElementById, getElementsByTagName и getElementsByName.

Более мощные способы поиска предлагают javascript-библиотеки.

Поиск по id

Самый удобный способ найти элемент в DOM - это получить его по id. Для этого используется вызов **document.getElementById(id)**

Например, следующий код изменит цвет текста на голубой в div'e c id="dataKeeper":

document.getElementById('dataKeeper').style.color = 'blue'

Поиск по тегу

Следующий способ - это получить все элементы с определенным тегом, и среди них искать нужный. Для этого служит document.getElementsByTagName(tag). Она возвращает массив из элементов, имеющих такой тег.

Например, можно получить второй элемент(нумерация в массиве идет с нуля) с тэгом li:

document.getElementsByTagName('LI')[1]

Что интересно, getElementsByTagName можно вызывать не только для document, но и вообще для любого элемента, у которого есть тег (не текстового).

При этом будут найдены только те объекты, которые находятся под этим элементом.

Например, следующий вызов получает список элементов LI, находящихся внутри первого тега div:

document.getElementsByTagName('DIV')[0].getElementsByTagName('LI')

Получить всех потомков

Вызов elem.getElementsByTagName('*') вернет список из всех детей узла elem в порядке их обхода.

Например, на таком DOM:

```
<div id="d1">
<ol id="ol1">
```


Библиотека ГГТУ им. П.О. Сухого

```
<li id="li1">1</li>
<li id="li2">2</li>
</ol>
</div>
```

Такой код:

```
var div = document.getElementById('d1')
var elems = div.getElementsByTagName('*')

for(var i=0; i<elems.length; i++) alert(elems[i].id)
```

Выведет последовательность: ol1, li1, li2.

Поиск по name: getElementsByName

Метод document.getElementsByName(name) возвращает все элементы, у которых имя (атрибут name) равно данному.

Он работает только с теми элементами, для которых в спецификации явно предусмотрен атрибут name: это form, input, a, select, textarea и ряд других, более редких.

Метод document.getElementsByName не будет работать с остальными элементами типа div,p и т.п.

Другие способы

Существуют и другие способы поиска по DOM: XPath, cssQuery и т.п. Как правило, они реализуются javascript-библиотеками для расширения стандартных возможностей браузеров.

Также есть метод getElementsByClassName для поиска элементов по классу, но он совсем не работает в IE, поэтому в чистом виде им никто не пользуется.

DOM: свойства VS атрибуты

У DOM-элементов в javascript есть **свойства** и **атрибуты**. И те и другие имеют имя и значение. Поэтому иногда разработчики путают одно с другим.

Между ними есть соответствие, но оно не однозначное и его лучше понимать.

12.2. СВОЙСТВА DOM-ЭЛЕМЕНТОВ

Узлы DOM являются объектами с точки зрения javascript. А у объектов есть свойства. Поэтому любому узлу можно назначить свойство, используя обычный синтаксис.

Библиотека БГТУ им. П.О. Сухого

```
var elem = document.getElementById('MyElement')
elem.mySuperProperty = 5
```

Значением свойства может быть любой объект. Это же javascript.

```
elem.master = {
name: vasya
}
alert(elem.master.name)
```

DOM-Атрибуты

Теперь посмотрим на DOM-элемент с другой стороны. Являясь элементом HTML, DOM-элемент может иметь любое количество атрибутов.

В следующем примере элемент имеет атрибуты id, class и нестандартный (валидатор будет ругаться) атрибут alpha.

<div id="MyElement" class="big" alpha="omega"></div>

Атрибуты можно добавлять, удалять и изменять. Для этого есть специальные методы:

setAttribute(name, value)	Устанавливает значение атрибута
getAttribute(name)	Получить значение атрибута
hasAttribute(name)	Проверить, есть ли такой атрибут
removeAttribute(name)	Удалить атрибут

Имя атрибута является регистронезависимым.

```
// название маленькими буквами
document.body.setAttribute('test', 123)

// большими буквами
document.body.getAttribute('TEST') // 123
```

Значением атрибута может быть только строка. Это же HTML..

DOM-атрибуты VS DOM-свойства

Все, вроде бы, ясно. Есть свойства. Есть атрибуты.

Но создатели javascript решили (с лучшими намерениями) запутать ситуацию и создать искусственное соответствие между свойством и атрибутом.

Синхронизация

А именно, браузер синхронизирует значения ряда свойств с атрибутами. Если меняется атрибут, то меняется и свойство с этим именем. И наоборот.

Библиотека БГУИР

Например:

```
document.body.id = 5
alert(document.body.getAttribute('id'))
```

А теперь - наоборот

```
document.body.setAttribute('id', 'NewId')
alert(document.body.id)
```

Такая синхронизация гарантируется для всех основных стандартных атрибутов.
При этом атрибуту с именем class соответствует свойство className, т.к. ключевое слово class зарезервировано в javascript.

Для "левых" атрибутов браузер ничего не гарантирует

```
document.body.setAttribute('cool', 'SomeValue')
alert(document.body.cool) // undefined везде кроме IE (почему - см ниже)
```

Возможные значения

Название атрибута не зависит от регистра

Атрибуты с именами "abc" и "ABC" - один и тот же атрибут.

```
document.body.setAttribute('abc', 1)
document.body.setAttribute('ABC', 5)
alert(document.body.getAttribute('abc')) // => стало 5
```

Но свойства в разных регистрах - два разных свойства.

```
document.body.abc = 1
document.body.ABC = 5
alert(document.body.abc) // => все еще 1
```

Увы, в Internet Explorer версии до 8.0 с этим проблемы. Этот браузер старается по возможности уравнять свойства и атрибуты. Но как быть, если свойства - регистрозависимы, а атрибуты - нет?

Создатели IE поступили хитро: setAttribute ставит оба свойства (abc и ABC).. Ну а getAttribute возвращает первое попавшееся из них, с учетом регистронезависимости. Если таких свойств несколько, то невозможно сказать, какое именно он вернет.

```
document.body.setAttribute('abc', 1)
document.body.setAttribute('ABC', 5)


// IE пытается уравнять свойства и атрибуты
alert(document.body.abc) // => 1
alert(document.body.ABC) // => 5
```

```
// но getAttribute выбирает первое попавшееся свойство
// за вычетом регистра букв
alert(document.body.getAttribute('abc')) // => 1
alert(document.body.getAttribute('ABC')) // => 1
```

Запустите этот пример в IE6/7 и, например, в Firefox, чтобы лучше понять различия.

Атрибут можно установить любой, а свойство - нет

Например, можно установить для тэга <body> *атрибут* tagName, но соответствующее *свойство* - только для чтения, поэтому оно не изменится:

```
document.body.setAttribute('tagName',1)
document.body.getAttribute('tagName') // 1
document.body.tagName // "BODY"
```

Вообще говоря, браузер не гарантирует синхронизацию атрибута и свойства.

Атрибуты и обработчики событий

В IE текстовое значение, присвоенное атрибуту onclick, не является функцией и не будет работать:

```
elem.setAttribute('onclick', 'alert(something)') // в IE не работает
```

Firefox корректно преобразовывает строку в функцию, поэтому там этот фрагмент работать будет. Но, вообще говоря, никакой браузер не обязан этого делать.

Атрибут - это строка (кроме IE)

Следующий код это наглядно демонстрирует:

```
document.body.setAttribute('v',{a:5})
alert(document.body.getAttribute('v')) // "[object Object]" в Firefox
alert(document.body.getAttribute('v').a) // 5 в IE
```

В первой строке атрибуту присвоено значение-объект. Firefox тут же автоматически преобразовал его в строку. А IE, в нарушение стандартов, оставил атрибут объектом.

Исключение className

Как уже говорилось в разделе о синхронизации, атрибуту class соответствует свойство className. Так получилось из-за того, что class является зарезервированным словом javascript.

Библиотека ГГТУ им. П.О.Сухого

В IE также является исключением атрибут for, для него используется свойство forHtml.

Свойство элемента и атрибут - это разные вещи. Не используйте одно вместо другого.

Вообще, обычно свойств хватает с головой.

12.3. ИЗМЕНЕНИЕ СТРАНИЦЫ ПОСРЕДСТВОМ DOM

Рассмотрим основные способы изменять DOM, вначале - в общих чертах, затем - на конкретном примере из жизни.

Создание и добавление элементов

Чтобы создать новый элемент - используется метод document.createElement(тип).

var newDiv = document.createElement('div')

Тут же можно и проставить свойства и содержание созданному элементу.

```
var newDiv = document.createElement('div')
newDiv.className = 'my-class'
newDiv.id = 'my-id'
newDiv.style.backgroundColor = 'red'

newDiv.innerHTML = 'Привет, мир!'
```

Добавление в DOM

Добавить новый элемент к детям существующего элемента можно методом appendChild, который в DOM есть у любого тега.

Код из следующего примера добавляет новые элементы к списку:

```
<ul id="list">
<li>Первый элемент</li>
</ul>
```

Список:

- Первый элемент

```
// элемент-список UL
var list = document.getElementById('list')

// новый элемент
var li = document.createElement('LI')
```


Библиотека ГГТУ им.П.О.Сухого

```
li.innerHTML = 'Новый элемент списка'
```

// добавление в конец

list.appendChild(li)

Метод appendChild всегда добавляет элемент последним в список детей.

Добавление в конкретное место

Новый элемент можно добавить не в конец детей, а перед нужным элементом.

Для этого используется метод insertBefore родительского элемента.

Он работает так же, как и appendChild, но принимает вторым параметром элемент, перед которым нужно вставлять.

parentElem.insertBefore(newElem, target)

Например, в том же списке добавим элементы перед первым li.

```
<ul id="list2">
<li>Первый элемент</li>
</ul>
```

- Первый элемент

```
// родительский элемент UL
var list = document.getElementById('list2')
// элемент для вставки перед ним (первый LI)
var firstLi = list.getElementsByTagName('LI')[0]

// новый элемент
var newListElem = document.createElement('LI')
newListElem.innerHTML = 'Новый элемент списка'

// вставка
list.insertBefore(newListElem, firstLi)
```

Метод insertBefore позволяет вставлять элемент в любое место, кроме как в конец. А с этим справляется appendChild. Так что эти методы дополняют друг друга.

Метода insertAfter нет, но нужную функцию легко написать на основе комбинации insertBefore и appendChild.

Удаление узла DOM

Чтобы убрать узел из документа - достаточно вызвать метод removeChild из его родителя.

list.removeChild(elem)

Библиотека ГГТУ им. П.О.Сухого

Если родителя нет "на руках", то обычно используют parentNode. Получается так:

elem.parentNode.removeChild(elem)

Неуклюже, но работает.

Пример - показ сообщения

Сделаем что-нибудь посложнее.

В качестве реального примера рассмотрим добавление сообщения на страницу. Чтобы показывалось посередине экрана и было красивее, чем обычный alert.

Сообщение в HTML-варианте (как обычно, можно посмотреть, нажав кнопку):

```
<style>
.my-message {
   width:300px;
   height:110px;
   background-color:#F08080;
   text-align: center;
   border: 2px groove black;
}
.my-message-title {
   height:20px;
   font-size:120%;
   background-color:red;
}
.my-message-body {
   padding: 5px;
   height: 50px;
}
</style>

<div class="my-message">
<div class="my-message-title">Заголовок</div>
<div class="my-message-body">Текст Сообщения</div>
<input class="my-message-ok" type="button" value="OK"/>
</div>
```


Библиотека ГГТУ им. П.О. Сухого

Как видно - сообщение вложено в DIV фиксированного размера my-message и состоит из заголовка my-message-title, тела my-message-body и кнопки OK, которая нужна, чтобы сообщение закрыть.

Кроме того, добавлено немного простых стилей, чтобы как-то смотрелось.

Показ сообщения состоит из нескольких этапов.

1. Создание DOM-элементов для показа сообщения
2. Позиционирование на экране
3. Запуск функции удаления сообщения по клику на OK

Создание

Для создания сколько-нибудь сложных структур DOM, как правило, используют либо готовые шаблоны и метод cloneNode, создающий копию узла, либо свойство innerHTML.

Следующая функция создает сообщение с указанным телом и заголовком.

```
function createMessage(title, body) {
// (1)
var container = document.createElement('div')

// (2)
container.innerHTML = '<div class="my-message"> \
<div class="my-message-title">'+title+'</div> \
<div class="my-message-body">'+body+'</div> \
<input class="my-message-ok" type="button" value="OK"/> \
</div>'

// (3)
return container.firstChild
}
```

Как видно, она поступает довольно хитро. Чтобы создать элемент по текстовому шаблону, она сначала создает временный элемент (1), а потом записывает (2) его как innerHTML временного элемента (1). Теперь готовый элемент можно получить и вернуть (3).

Позиционирование

Сообщение будем позиционировать абсолютно, в центре по ширине и на 200 пикселов ниже верхней границы экрана.

Библиотека ГГТУ им. П.О.Сухого

```
function positionMessage(elem) {
    elem.style.position = 'absolute';
    var scroll = document.documentElement.scrollTop ||
document.body.scrollTop;
    elem.style.top = scroll + 200 + 'px';
    elem.style.left = Math.floor(document.body.clientWidth/2) - 150 + 'px';
}
```

Не вдаваясь в тонкости позиционирования - заметим, что для свойства top 200 пикселов прибавляются к текущей вертикальной прокрутке, которую браузер отсчитывает либо от documentElement либо от body - зависит от DOCTYPE и типа браузера.

При установке left от центра экрана вычитается половина ширины DIV'а с сообщением (у него стоит width:300).

Закрытие

Наконец, следующая функция вешает на кнопку OK функцию, удаляющую элемент с сообщением из DOM.

```
function addCloseOnClick(messageElem) {
    var input = messageElem.getElementsByTagName('INPUT')[0]
    input.onclick = function() {
        messageElem.parentNode.removeChild(messageElem)
    }
}
```

Обратите внимание, при получении элемента функции не используют DOM-свойства previousSibling/nextSibling.

Этому есть две причины. Первая - надежность. Мы можем изменить шаблон сообщения, вставить дополнительный элемент - и ничего не должно сломаться.
Вторая - это наличие текстовых элементов. Свойства previousSibling/nextSibling будут перечислять их наравне с остальными элементами, и придется их отфильтровывать.

Запуск

Создадим одну функцию, которая выполняет все необходимые для показа сообщения операции.

```
function setupMessageButton(title, body) {
    // создать
```


Библиотека БГУИР

```
    var messageElem = createMessage(title, body)
    // позиционировать
    positionMessage(messageElem)
    // добавить обработчик на закрытие
    addCloseOnClick(messageElem)
    // вставить в документ
    document.body.appendChild(messageElem)
}
```

Протестировать то, что получилось, нам поможет следующая кнопка:

```
<input type="button" value="Показать"
        onclick="setupMessageButton('Привет!', 'Ваше сообщение')"/>
```


Библиотека ГГТУ им.П.О.Сухого

Рябченко Алексей Иванович
Родионов Андрей Александрович
Горский Сергей Михайлович

ПРОЕКТИРОВАНИЕ ДИНАМИЧЕСКИХ СТРАНИЦ

Курс лекций
по одноименной дисциплине
для слушателей специальности 1-40 01 74
«Web-дизайн и компьютерная графика»
заочной формы обучения

Подписано к размещению в электронную библиотеку
ГГТУ им. П. О. Сухого в качестве электронного
учебно-методического документа 10.12.12.

Рег. № 48Е.
http://www.gstu.by

Библиотека ГГТУ им.П.О.Сухого