Джо Барнс

Практикум по программированию на JavaScript

ИНТУИТ
НАЦИОНАЛЬНЫЙ ОТКРЫТЫЙ УНИВЕРСИТЕТ

Практикум по программированию на JavaScript

2-е издание, исправленное

Джо Барнс

Национальный Открытый Университет "ИНТУИТ"
2016

Практикум по программированию на JavaScript/ Джо Барнс - М.: Национальный Открытый Университет "ИНТУИТ", 2016

Цель этих 30 уроков состоит в том, чтобы научить читателя самостоятельно создавать сценарии JavaScript.
Если вы пытались ознакомиться с JavaScript с помощью книг или Интернет, то почти наверняка это оказалось не просто. Изучение JavaScript является, по сути, изучением нового языка. Текст может казаться знакомым, но конструкции предложений совершенно другие.

JavaScript за 30 шагов

Обращение к читателю.

Введение

Цель этих 30 уроков состоит в том, чтобы научить читателя самостоятельно создавать сценарии JavaScript. Если вы пытались ознакомиться с JavaScript с помощью книг или Интернет, то почти наверняка это оказалось не просто. Изучение JavaScript является, по сути, изучением нового языка. Текст может казаться знакомым, но конструкции предложений совершенно другие.

В интернет существует множество готовых сценариев (например, на сайте javascripts.com), но для их использования все равно необходимо знание JavaScript. Существует много исследований, показывающих, что изучение с помощью лекций или чтения книг малоэффективно. Мы собираемся подойти к изучению JavaScript по-другому. Лучше всего изучать этот материал в классе с преподавателем, который может помочь выполнить проект и объяснит сложные моменты. К сожалению, данная книга все еще остается односторонним общением. Один умный человек сказал как-то: "Скажи мне, и я забуду. Покажи мне, и я запомню. Вовлеки меня, и я научусь." Мы попытаемся вас вовлечь.

Каждый из следующих 30 уроков рассматривает один сценарий JavaScript и расщепляет его на составные части, чтобы можно было увидеть, как он работает. Наша задача: не просто показать, что нечто работает, а доказательно объяснить, почему.

Структура урока

Всего имеется 30 уроков. Каждый использует одну и ту же структуру.

Урок начинается с краткого объяснения того, что делает рассматриваемый сценарий и в чем его познавательная ценность для читателя.

Затем следует сценарий в текстовой форме.

На третьем шаге мы видим результат деятельности сценария.

После этого мы разбираем сценарий на составные блоки, используемые для создания целого.

Наконец, каждый урок имеет упражнение. Здесь читателю предлагается немного изменить разобранный сценарий. Возможно, что вам будет нужно изменить сценарий таким образом, чтобы он порождал ошибки. В любом случае надо будет создать 30 новых сценариев из 30 данных.

Давайте начнем

Будьте внимательны при изучении этих уроков. Студенты часто хотят преодолеть учебник как можно быстрее. Обычно при таком подходе все команды перемешиваются в голове. Мозгу нужно время, чтобы усвоить новый материал. Лучше всего — не старайтесь делать больше двух уроков в день.

Студенты часто говорят, что прочитали всю главу, но не могут запомнить, что прочитали. Это потому, что целью было добраться до конца главы, а не получить максимум знаний от чтения. Скорость убивает.

Вы ворочались, прежде чем начали ползать, прежде чем начали ходить или бегать. Дайте голове время немного поворочаться вокруг простых сценариев.

Основные понятия

Введение в JavaScript. Размещение сценария на Web-странице.

Концепция

Первый сценарий предназначен для знакомства с основами создания и размещения JavaScript на Web-странице. В ходе урока вы узнаете о том, что можно и чего нельзя делать в JavaScript. Этот сценарий выводит текст на Web-страницу. В данном случае текст будет красного цвета. Итак:

Сценарий

```
<SCRIPT LANGUAGE="javascript">
document.write
("<FONT COLOR='RED'>Это красный текст </FONT>")
</SCRIPT>
```

Результат работы сценария

Это красный текст

Разбор сценария

Так как это очень простой сценарий, то поговорим сначала о том, что такое JavaScript вообще.

Что такое JavaScript?

Во-первых, это не Java. Тут легко запутаться и решить, что Java и JavaScript — одно и то же. Это не так. Java — язык программирования, разработанный в компании Sun Microsystems. А JavaScript придумали в компании Netscape. Но это не единственное отличие.

Оба языка схожи в том, что представляют собой OOP (Object Orientated

Programming, объектно-ориентированный язык программирования). Это значит, что с их помощью можно строить небольшие объекты, из которых потом складывается целое. Это станет понятнее по ходу дела. Главное отличие в том, что Java позволяет создавать совершенно самостоятельные события. "Java-applet" ("приложеньице" или просто апплет, называемый так, поскольку является небольшим приложением) может запускаться на Web-странице, но на самом деле это полностью независимая самодостаточная программа, хотя и маленькая. К тому же, ее нельзя выполнить в виде текста. Для запуска ее необходимо "транслировать" в то, что называется "машинным языком".

Netscape как бы упростил Java до набора более простых команд. JavaScript не может существовать сам по себе, он должен выполняться внутри Web-страницы, а Web-страница должна просматриваться в браузере, который понимает язык JavaScript (скажем, Netscape Communicator или Internet Explorer).

Как создать сценарий JavaScript

Прежде всего надо помнить, что JavaScript — это не HTML! Часто задают вопрос, не является ли первый просто другой версией второго. Нет. Однако у JavaScript и HTML имеются очень похожие правила.

Во-первых, JavaScript располагается внутри документа HTML. Где именно, мы обсудим позже. JavaScript сохраняется в виде текста вместе с документом HTML.

Главная же разница состоит в том, что HTML многое прощает с точки зрения своей формы записи. Не имеет значения, сколько пробелов вы оставляете между словами или абзацами. По правде говоря, HTML можно было бы писать одной сплошной строкой.

Совсем другое дело JavaScript. У него четкая форма. И пренебрегать ею можно лишь изредка. Например, вторая строка сценария этого урока выглядит следующим образом.

```
document.write
("<FONT COLOR='RED'>Это красный текст</font>")
```

Она целиком находится на одной линии и должна сохранять свою форму. Предположим, вы скопировали ее в текстовый редактор с узкими страницами, и поля разорвали строку:

```
document.write
("<FONT COLOR='RED'>Это красный текст</font>
")
```

Вы изменили форму, и в сценарий вкралась ошибка (в следующем уроке мы поговорим об ошибках и о том, как их исправлять).

Редактирование JavaScript

Пишете вы сценарий или редактируете, не давайте полям страницы вставать у вас на пути. Всегда используйте для работы текстовой редактор без полей. Речь идет не просто о широкой странице. Имеется в виду ПОЛНОЕ ОТСУТСТВИЕ ПОЛЕЙ. У вас должна быть возможность написать строку длиной в несколько километров. Иначе не оберетесь проблем.

Важен ли РеГиСтр символов для JavaScript?

Да.

Вернемся к разбору сценария

Начнем с первой строки:

```
<SCRIPT LANGUAGE="JavaScript">
```

Это код HTML, который дает браузеру понять, что с этого места начинается JavaScript. Кажется, это несложно. Любой сценарий JavaScript начинается с такой команды.

А как насчет части `LANGUAGE` (язык) `="JavaScript"`? Разве это необходимо? Да. Есть еще и другие типы сценариев, например, VBS или LiveScript. Так что команда `LANGUAGE` не даст браузеру запутаться.

Так как у нас всего три строчки текста, то можно сразу заглянуть в конец. Код

```
</SCRIPT>
```

заканчивает любой сценарий JavaScript без исключений. Запомните это навсегда, потому что к этому вопросу мы больше возвращаться не будем. Начинаем со `<SCRIPT LANGUAGE="javascript">` и заканчиваем `</SCRIPT>`. Двигаемся дальше...

Вот основная часть сценария:

```
document.write("<FONT COLOR='RED'>Это красный текст</FONT>")
```

Не так уж трудно догадаться, что означает каждая часть сценария, но чтобы в дальнейшем пользоваться общими терминами, рассмотрим его подробно.

Сценарий устроен следующим образом. С помощью `DOCUMENT` объявляется документ (документ HTML). Этот документ будет изменен — в нем что-то будет написано (`WRITE`). То, что будет написано, находится внутри скобок.

Настала очередь терминов. `DOCUMENT` представляет собой объект. Слово `WRITE` (писать), отделенное точкой, называется методом объекта. Таким образом, сценарий по сути говорит: "Возьмите объект (что-то, уже существующее) и что-то в нем запишите".

Обратите внимание, что текст внутри скобок находится в кавычках. В HTML эти кавычки не требуются. Здесь они необходимы. Никогда нельзя про них забывать.

Текст в кавычках представляет собой простой код HTML. Легко увидеть в нем команду `<FONT>`, которая делает текст красным. Обратите внимание, что RED находится в одинарных кавычках. Если использовать двойные, JavaScript решит, что это конец строки, и получится, что только часть текста будет записана в объект, а это уже ошибка.

Запомните: внутри двойных кавычек используются одинарные.

Получается, что JavaScript перекрасил текст в красный цвет? Нет, это сделал HTML. А JavaScript только записал код на страницу.

Задание

Измените сценарий так, чтобы выводились две строки текста, красная и синяя. Но это надо сделать с помощью дополнительных команд JavaScript, а не просто добавить код HTML к приведенному примеру. На странице должно выводиться следующее:

Это красный текст
Это синий текст

Возможное решение

... было создано следующим кодом:

```
<SCRIPT type="text/javascript">
 document.write
 ("<FONT COLOR='RED'>Это красный текст</FONT><BR>")
 document.write
 ("<FONT COLOR='BLUE'>Это синий текст</FONT>")
</SCRIPT>
```

Результат получен добавлением в сценарий второй строки `document.write` и изменением кода HTML внутри экземпляра (параметра) метода. Добавлена также команда `<BR>` в конце первого экземпляра метода, чтобы получить две строки.

Сообщения об ошибках

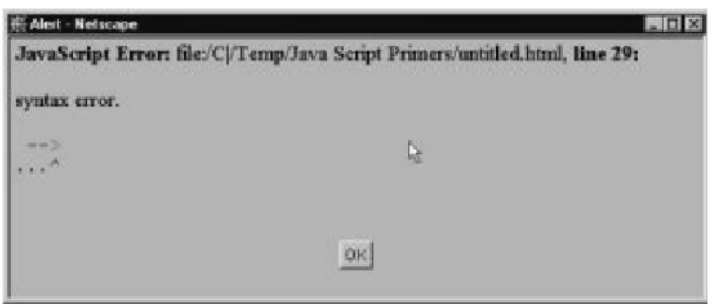
Ошибки в сценариях. Виды ошибок и методы их поиска.

Концепция

В большинстве книг по JavaScript почти не рассматриваются сообщения об ошибках.

Видимо, авторы предполагают, что все делается правильно с самого первого раза и такие сообщения никогда не появляются. Добро пожаловать в реальный мир!

Если вы хотя бы раз пытались написать сценарий JavaScript или использовать на Web-странице готовый, то вам должно быть знакомо это чувство, которое возникает, когда вы считаете, что уже все в порядке и тут... бац! Выскакивает такое окошко:



Изображение сообщения об ошибке

В этом учебнике мы расскажем, что делать, когда возникают сообщения об ошибках. Когда вы начнете писать сценарии JavaScript, они будут появляться постоянно, так как являются неотъемлемой частью процесса создания.

Пожалуйста, запомните! В более поздних версиях MSIE и Navigator это окошко может не появляться.

При использовании MSIE сообщение об ошибке появится сначала как треугольный значок в нижнем левом углу. В этом треугольнике будет находиться восклицательный знак. Будет также присутствовать текст,

сообщающий, что на странице встретились ошибки. Щелкните на этом треугольнике, чтобы получить сообщение об ошибке, которое рассматривается в данном учебнике. Или, если вы хотите, чтобы окно ошибки выводилось сразу, без щелчка на значке, перейдите в меню Tools (Сервис) и выберите Internet Options (Свойства обозревателя). В Internet Options щелкните на вкладке Advanced (Дополнительно) и поставьте флажок у строки "Display a notification about every script error" (Показывать уведомление о каждой ошибке сценария).

Затем, в зависимости от конфигурации системы, вы сможете получить окно ошибки. В окне ошибки существует также кнопка "Details" (Подробнее), при нажатии на которую выводится текстовое описание ошибки.

При использовании более поздней версии Netscape Navigator в строке состояния выводятся указания пользователю. При возникновении ошибки будет предложено ввести `javascript:` в строке адреса. Затем будет выведена ошибка и соответствующее текстовое описание.

Сообщение об ошибке

В основном ошибки бывают двух типов: синтаксиса и времени выполнения. Ошибка синтаксиса означает опечатку или неправильную конфигурацию JavaScript. Ошибка времени выполнения означает, что была использована неправильная команда. В любом случае получается ошибка. Где-то что-то было перепутано.

Существуют программы, которые помогают исправлять ошибки и выполнять так называемый процесс отладки ("debugging"), но все можно сделать вручную. На самом деле это даже легче, чем можно подумать.

Исправление ошибок

Говорят, что наилучший способ исправить ошибку — это ее не совершать, но сказать легче, чем сделать. Тем не менее можно свести количество ошибок к минимуму, пользуясь текстовым редактором без полей. Кроме того, отводите каждой команде JavaScript отдельную строку. Ни к чему разбивать длинные строки на несколько коротких. Это

само по себе может привести к ошибке. И все же, можно поспорить, что каждый раз, принимаясь за создание сценариев, вы будете получать такие сообщения. Так что давайте разберемся, как их устранять.

В этих всплывающих окошках сообщений об ошибке есть одна замечательная вещь: они сами говорят, где и в чем состоит проблема. Взгляните на сообщение. У нас синтаксическая ошибка, означающая неправильную конфигурацию сценария, и находится она на строке 29. Более того, сообщение об ошибке прямо указывает на проблемную область. Было бы неплохо иметь такое и в HTML?

Строка ошибки

Когда сообщение об ошибке указывает на строку ошибки, то строку с ошибкой нужно отсчитывать от самого верха документа HTML, а не от первой строки JavaScript. Например, в приведенном ниже документе допущена ошибка в строке 9. Это ошибка синтаксиса, так как экземпляр (параметра) не заканчивается на той же строке, где начался. Видите, как скобка перескочила на следующую строчку?

```
<HTML>
<HEAD>
<TITLE></TITLE>
</HEAD>
<BODY>

<SCRIPT LANGUAGE="javascript">
document.write("текст для страницы
"
)
</SCRIPT>
</BODY>
</HTML>
```

Но почему ошибка на строке 9, а не 8? Потому, что счет начинается с верхнего края документа HTML, не пропуская ни одной строки. Вот этот документ еще раз с пронумерованными строками.

```
(строка 1)  <HTML>
```

```
(строка 2)  <HEAD>
(строка 3)  <TITLE></TITLE>
(строка 4)  </HEAD>
(строка 5)  <BODY>
(строка 6)
(строка 7)  <SCRIPT LANGUAGE="javascript">
(строка 8)  document.write("text for the page"
(строка 9)  )
(строка 10) </SCRIPT>
(строка 11) </BODY>
(строка 12) </HTML>
```

Так что считайте все строки, даже пустые.

Что дальше?

Как только строка с ошибкой найдена, необходимо решить, что делать. Чаще всего это будет синтаксическая ошибка — либо разорванная строка, либо опечатка, либо двойные кавычки вместо одинарных (несбалансированные кавычки) и тому подобное.

Если это ошибка времени выполнения, значит, команда, на которую указывает сообщение, не укладывается в логическую последовательность. Например, сценарий вызывает кнопку с помощью команды, которая на самом деле вызывает текстовое поле.

Множественные ошибки

Ничто не раздражает больше, чем многократные сообщения об ошибках. Можно только сидеть и смотреть, как на экране плодятся окна. Можно решить, что многократные окна сообщений означают многократные ошибки. Не всегда.

JavaScript — это чрезвычайно логичный язык, требующий, чтобы все шло своим чередом, в линейной последовательности. Допустим, имеется 10 ошибок в длинном сценарии. Сообщения накладываются одно на другое, и последняя обнаруженная компьютером ошибка окажется сверху. Не пытайтесь сразу ее исправлять, возможно, в

действительности ее даже не существует.

Может случиться так, что первая ошибка сценария вызывает все остальные. Так что исправлять ошибки следует последовательно с начала документа HTML. Сколько раз бывало так, что выскакивает 20 окон с ошибками, а разрешить все проблемы удается исправлением одной единственной первой ошибки!

Поэтому исправлять ошибки надо по одной от начала до конца. И каждый раз, исправив одну ошибку, запускайте сценарий. Можно получить 20 сообщений об ошибках, а исправлять придется одну или две.

Отсутствие определения

Эта ошибка времени выполнения также достаточно распространена. Она означает, что в сценарии что-то не согласовано. Надо следить за тем, чтобы строка кода не появлялась в коде слишком рано, когда используемые в ней переменные еще не определены. Если дело не в этом, попробуйте стереть строку с ошибкой, ведь ее всегда можно вернуть на место. Случаются и банальные опечатки. Приглядитесь к тексту повнимательнее, опечатки случаются чаще, чем можно себе представить.

Вот практически все, что на данном этапе можно сказать про ошибки. Правда, этих знаний хватит, чтобы исправить 99% возникающих неприятностей. Просто помните, что сообщение — это на самом деле плюс. Без них пришлось бы сидеть, тупо уставясь в пустую страницу, не имея ни малейшего понятия, в чем проблема. Они весьма полезны, если взглянуть на них с правильной точки зрения.

Задание

Ниже показан сценарий с ошибкой. Загружая страницу с этим сценарием, браузер должен выдать два сообщения об ошибке. Исправьте их. Некоторые команды покажутся незнакомыми, но это неважно. Сообщения содержат достаточно информации, чтобы исправить сценарий.

Если сценарий выполнится правильно, то на странице появится текущая дата.

Подсказка: возможно, сначала будет получено только одно сообщение. Вторая ошибка появится, когда будет исправлена первая.

Код с ошибкой

```
<SCRIPT type="text/javascript">
...x
dothis = new Date()
month = dothis.getMonth()
month = (month * 1) + 1
day = dothis.getDate()
year = dothis.getFullYear()
document.wrte(" ",month,"/",day,"/",year," ")
</SCRIPT>
```

Возможное решение

Так действует скрипт — показывает дату. Чтобы заставить его работать, надо исправить две ошибки. Первой должна быть синтаксическая ошибка, утверждающая, что `...x` не определено. Проще всего исправить эту ошибку будет удалением текста. Она является просто шуткой.

При повторном запуске сценария должна появиться другая синтаксическая ошибка, утверждающая, что `document.wrte` не является функцией. Причина ошибки — опечатка в слове `write`.

Дата и время

Функции для работы с датой и временем.

Концепция

Из Урока 1 известно, что существует такой объект, как документ. Иначе в нем ничего нельзя было бы написать.

В этом уроке будут рассмотрены семь новых методов: `getDay()`, `getDate()`, `getMonth()`, `getYear()`, `getHours()`, `getMinutes()`, и `getSeconds()` (получить День, Число, Месяц, Год, Час, Минуту, Секунду). Все они уже существуют, их можно взять и поместить на Web-страницу. Проблема в том, что это всего лишь методы. Для действия им нужен объект, а документ для этих целей не годится... так что необходимо создать объект.

Сценарий

```
<SCRIPT LANGUAGE="JavaScript">

//Сценарий выводит точную дату и время посещения страницы

Now = new Date();
document.write("Сегодня " + Now.getDate()+
"-" + (Now.getMonth()+1) + "-" + Now.getFullYear()
+ ". Вы зашли на Web-страницу ровно в: " + Now.getHours() +
":" + Now.getMinutes() + " и " + Now.getSeconds() +
" секунд.")

</SCRIPT>
```

Кстати, строка `document.write` не должна прерываться. Она разбита на несколько строк, чтобы удобнее было читать.

Результат работы сценария

Сегодня 18-7-2006. Вы зашли на эту Web-страницу ровно в: 8:42 и 3 сек

Разбор сценария

Не пугайтесь размера сценария, скоро все будет понятно. Для начала еще раз напомним о формате сценария. Строка `document.write` уходит далеко за границы экрана. Эту форму необходимо сохранить. Если разбить эту строку на две, браузер выдаст сообщение об ошибке.

Что означает // ?

Постепенно, по мере чтения учебника вы будете узнавать все больше новых команд. Это одна из них. Двойная косая черта указывает на комментарий внутри сценария. Она означает, что следующий за ней текст не будет использоваться в процессе. Количество строк комментария ничем не ограничено, надо только помнить, что каждая строка должна начинаться с двойной косой черты //.

Методы Date (Дата) и Time (Время)

Если посмотреть сценарий, то можно видеть, что результат создается командой записи в документ числа, месяца, года, часа, минуты и секунды. Несколько дополнительных слов объясняют, что вы видите.

Каждый из этих объектов был создан с помощью метода в виде `getЧто-либо()`. Обратите внимание на заглавную букву. Сначала "get" в нижнем регистре, потом слово с Заглавной буквы, описывающее объект.

Первое: запомните, что каждый из этих объектов является числом. Все методы возвращают только числа. Даже метод `getDay()`, который возвращает день недели, выражается числом от единицы до семи.

Начнем с месяца. Как уже говорилось ранее, `getMonth()` — это метод, отвечающий за месяц. Теперь задумаемся о том, на какой объект воздействует метод `getMonth()`. Т.е. теперь необходимо определиться, методом какого объекта является `getMonth()`.

(Помните из Урока 1? Метод действует на объект).

Может показаться, что `getЧто-либо()` — это метод объекта `document`. Вовсе нет. Метод документа — `write`. `getMonth()` на самом деле является методом объекта `Date`. Взгляните на сценарий еще раз. Объект `Date` задается в команде:

```
Now = new Date();
```

С этим объектом будет работать метод `getMonth()`. Имея дело с датой и временем, всякий раз пользуйтесь той же схемой. Прежде всего необходимо создать объект. В данном случае объект называется `Now` (сейчас). Видите? Между прочим, я мог бы назвать его `Zork` или `Fred`, браузеру все равно. Это не имеет значения, если объект получает оригинальное имя, которое больше нигде в JavaScript не встречается.

Если вам кажется, что кое-что тут поставлено с ног на голову, это легко понять. Такое впечатление, что следует писать `new Date = Now`, но это не так. Вы изучаете незнакомый язык и обязаны подчиняться его правилам.

Команда говорит: `Now` — это объект, который представляет `new Date()` (новую Дату). Дата обязательно должна быть новой. Таким способом вы будете получать новую дату каждый раз, когда заходите на страницу или обновляете ее. Без команды `new` дата будет оставаться статичной.

Обратите внимание и на точку с запятой в конце строки. Это действует как признак конца оператора. Она указывает на то, что строка JavaScript закончена. Без нее браузер решил бы, что команда продолжается на следующей строке. Ошибка.

Итак, у нас есть объект, на который может воздействовать метод `getMonth()`. Нам нужно напечатать месяц на странице, значит, где-то должна быть команда `document.write()`. Мы знаем также, что текст в скобках будет выведен на страницу, поэтому давайте напишем все это, следуя логике.

1. Сначала пишем `<SCRIPT LANGUAGE="javascript">`.

2. Затем вставляем комментарий о том, для чего предназначен сценарий.
3. Прежде чем можно будет обратиться к `getMonth()`, необходимо создать объект. Не забудьте поставить в конце строки точку с запятой.
4. Теперь можно поместить оператор `document.write`.
5. Текст в скобках после `document.write` оформляем по правилам Урока 1.
    - Текст, выводимый на странице, должен быть окружен двойными кавычками (одинарные кавычки для кода HTML внутри двойных).
    - Новое правило: сочетание текста и команд требует знака "плюс" `+` между элементами.
    - Объект и метод разделены точкой, так что команда "поместить месяц" должна выглядеть так: `Now.getMonth()`.
    - Новое правило: `Now.getMonth()` — это не текст, который должен быть виден на странице, а команда, которая указывает месяц. Поэтому не нужно ставить ее ни в какие кавычки. Обратите внимание, что к `getMonth` добавлена `1`: `(Now.getMonth()+1)`. Причина в том, что возвращаемые месяцы ведут отсчет от 0, поэтому для правильного вывода месяца необходимо добавлять `1`. Прибавление единицы помещено в скобки `(` и `)`, чтобы знак `+` не создавал путаницы со знаками `+`, соединяющими выводимые элементы.
6. Заканчиваем командой `</SCRIPT>`.

Вот что у нас получилось:

```
<SCRIPT LANGUAGE="javascript">

//Сценарий выведет на странице номер месяца

Now = new Date();
document.write("Сейчас месяц " + (Now.getMonth()+1))

</SCRIPT>
```

Посмотрите на полный сценарий еще раз. Длинная строка текста уже не кажется такой страшной. Это просто объект `Now` и следующий после него метод `getЧто-либо()`. Я разделил элементы даты дефисом. Помните, что дефис должен быть виден на странице, поэтому его следует ставить в кавычки. Все части связаны между собой знаком плюса `+`.

Добавление пробелов

Один небольшой прием-совет, который надо знать. Сколько бы пробелов вы не вставили до и после знака плюс, это никак не повлияет на видимый результат. Элементы пойдут сплошным текстом. Поэтому, если в текст требуется вставить пробелы, добавляйте их в части текста в кавычках. Например:

```
"Сейчас ровно   "
```

Видите, здесь добавлены два пробела перед второй кавычкой? Это превратится в два пробела на странице, когда сценарий выполнится. Помните: это не HTML. В Javascript существуют свои правила относительно пробелов.

Создание длинной строки

Мы не будем разбирать ее целиком, потому что вы наверняка уже поняли, что надо делать. Оставим только строку с датой. Вот как это выглядит:

```
document.write("Сегодня "
+  (Now.getMonth()+1)+
"-" + Now.getDate() + "-"
+ Now.getFullYear() + ".
Вы зашли на Web-страницу ровно в : "
+ Now.getHours() +
":" + Now.getMinutes()  + " и "
+ Now.getSeconds() +
" секунд")
```

1. Начинаем с "`Сегодня `", прибавив в конце пробел.
2. Затем следует знак плюс.
3. `(Now.getMonth() + 1)` добавляется без кавычек, потому что нам нужен не этот текст, а возвращаемое число.
4. Еще плюс.
5. Потом дефис в кавычках, чтобы отделить следующее число. Никаких пробелов, потому что они должны стоять вплотную.
6. Плюс.
7. Потом `Now.getDate()` без кавычек, чтобы у нас был день.
8. Плюс.
9. Еще дефис в кавычках, чтобы он был виден на странице.
10. Плюс.
11. Еще один метод `Now.getFullYear` сообщит год.

Продолжайте дальше по этой схеме, и сценарий выведет именно то, что вы хотите. Тогда вы всем сможете сказать, который час.

Известная проблема

Об этом уже говорилось ранее, но поговорим немного еще. Должно быть, вы заметили, что номер месяца будет на единицу меньше чем нужно, если использовать просто `getMonth()`. Почему? Вспомните, что числа сообщает JavaScript, а JavaScript любит считать от нуля. То есть, январь нулевой месяц и так далее.

Что же делать? Прибавить `1`, разумеется! Тут требуется известная сноровка. Нужно ввести несколько переменных, то есть присвоить имя некоему элементу (это еще будет в Уроке 6). Вы присваиваете `new Date()` имя, как уже делали раньше. Затем присваиваете имя коду, который вызывает месяц. Ниже я назвал его `mpo` (Месяц Плюс Один). И прибавляете к этому имени единицу. Эту новую команду я назвал `mpo1`. Звучит несколько запутанно, но не торопитесь. Вот как это выглядит:

```
<SCRIPT LANGUAGE="javascript">

RightNow = new Date();
var mpo = RightNow.getMonth();
```

```
var mpo1 = mpo + 1;
document.write("Сегодня месяц " +mpo1+ ".");

</SCRIPT>
```

Вот что получилось:

```
Сегодня месяц 8.
```

Это уж правильный месяц, или как?

Задание

Сегодняшнее задание не очень сложное. Напишите сценарий, который выводит на Web-странице дату, разделенную косой чертой. Приветственный текст должен быть зеленого цвета. Пока не беспокойтесь о числах, мы рассмотрим изменение их цвета позже. Добавьте также комментарий о том, что это вы написали этот сценарий.

Возможное решение

```
Привет! Сегодня у нас 18/8/2006.
```

Вот сценарий:

```
<SCRIPT LANGUAGE="JavaScript">

// Автор сценария tttt@mail.ru
Now = new Date();
var mpo = Now.getMonth();
var mpo1 = mpo + 1;
document.write("<FONT COLOR='green'>Привет! Сегодня у нас:</FONT>
+ Now.getDate() + "/"
+ mpo1 + "/"
+ Now.getFullYear() + ".");
</SCRIPT>
```

Обратите внимание на команды `FONT` и одинарные кавычки при указании зеленого цвета шрифта. Дату можно было скопировать из

первоначального сценария, заменив дефисы на косую черту.

Обработчики событий: onMouseOver

События в JavaScript. Обработка события onMouseOver.

Концепция

Мы обсудили объекты и методы. Теперь приступим к рассмотрению событий (events).

Сначала несколько вводных замечаний. События (event) и обработчики событий (event handler) относятся к JavaScript, но они скорее "встроены" в HTML-код, а не существуют самостоятельно, как те сценарии, которые мы создали. События являются встроенными, так что они не требуют команд `<SCRIPT>` и `</SCRIPT>`. Сами они являются не сценариями, а скорее небольшими интерфейсами, обеспечивающими взаимодействие между страницей и читателем.

События — это то, что происходит. Они добавят динамики Web-сайту. Увидев их, посетители сайта скажут: "Ух ты!", а длинные сценарии JavaScript для этого совсем не понадобятся.

Существует множество событий, с которыми мы со временем познакомимся, но для начала выберем одно из наиболее популярных — `onMouseOver` (навести курсор мыши).

Сценарий

```
<A HREF="http://www.INTUIT.ru"
onMouseOver="window.status='Почтовая служба';
return true">Ссылка</A>

<A HREF="http://www.INTUIT.ru" onMouseOver="document.bgColor='pink';
```

Результат работы сценария

При наведении курсора мыши на ссылку строка состояния в окне браузера изменится.

Разбор сценария

Вы уже знаете достаточно, чтобы понять смысл написанного. Давайте быстро разберем сценарий и попробуем его как-нибудь изменить.

Во-первых, `onMouseOver` (обратите внимание на заглавные буквы) представляет собой обработчик событий (Event Handler) гипертекстовой ссылки. Это понятно? Он используется внутри гиперссылки.

Формат гипертекстовой ссылки остается без изменений. Те же команды и те же двойные кавычки. Обработчик событий `onMouseOver` ставится сразу же после адреса URL. Видите?

Событие (Event) приводится в действие, когда браузер распознает `onMouseOver=""`. Общая схема уже должна быть немного понятна: два элемента, разделенные точкой. До сих пор это означало, что один является объектом, а другой методом. Но в данном случае объектом является `window` (окно), оно существует; `status` (статус) представляет собой свойство (property) окна. Это небольшой участок окна, где должен разместиться следующий далее в команде текст. Это проще запомнить, если представить, что метод обычно выражается глаголом, как `write` (писать) или `get` (получить). Свойство выражается существительным и существует как небольшая часть элемента, стоящего перед точкой.

Если у `window` есть изменяемое свойство `status`, значит, можно изменить и другие свойства окна. Это верно, но не будем забегать здесь вперед. Разберемся пока с окном и его строкой состояния.

После `window.status` ставится знак равенства `=`, за которым следует то, что должно произойти. В данном случае это текст в одинарных кавычках. Он появится в строке состояния, когда вы наведете курсор на гипертекстовую ссылку.

Пожалуйста, обратите еще раз внимание на точку с запятой в конце командной строки.

Затем следует `return true`. Эти два слова имеют вполне определенное влияние на то, что произойдет, когда указатель мыши

переместится на ссылку. Если они присутствуют, сценарий проверит, есть ли строка состояния. Если проверка будет успешной (`true`), происходит событие. Обратите внимание, что когда указатель мыши перемещается на ссылку, то текст в строке состояния блокируется. Он больше не изменяется при последующих перемещениях указателя мыши на ссылку. (Если обновить страницу, то можно будет это увидеть).

Что произойдет, если эти два слова потеряются? Давайте подумаем. Если проверка строки состояния отсутствует, то выполняется действие по умолчанию. По умолчанию в HTML выводится URL-ссылка, на которую указывает курсор мыши. Затем, когда курсор смещается со ссылки, произойдет событие. И так как проверка отсутствует, то событие будет происходить всякий раз при перемещении курсора мыши над ссылкой. Такой результат будет, наверно, лучше.

Другие свойства

Вернемся к свойствам, о которых мы говорили ранее. Если свойства есть у окна, то другие объекты тоже должны иметь свойства. Как насчет цвета фона? Это же свойство, не так ли? В HTML цветом фона страницы управляет команда `BGCOLOR`.

То же самое и здесь, только обязательно соблюдайте регистр символов. В JavaScript цвет фона обозначается `bgColor`. Подумаем, как создать ссылку, которая изменяла бы цвет фона окна с помощью события `onMouseOver`.

1. Во-первых, раз это ссылка, то сохраним тот же формат, который использовался ранее.
2. Что изменяется, окно или документ? Куда идет команда `bgColor`, когда создается Web-страница? В документ. Значит, это и есть нужный объект. Заменим `window` на `document`.
3. Мы собираемся изменить фоновый цвет объекта, потому заменим `status` на `bgColor`.
4. Появление текста нам больше не нужно, поэтому заменим его цветом. Возьмем розовый (`pink`).
5. Нам нужно, чтобы новый цвет оставался независимо от того, сколько раз мы будем наводить курсор на ссылку, потому

вставляем `return true` после точки с запятой.

Вот что получилось ...

```
<A HREF="http://www.INTUIT.ru"
onMouseOver="document.bgColor='pink';
return true">Щелкните здесь</A>
```

Если требуется и то, и другое

Подумаем, как можно это сделать. Здравый смысл подсказывает, что нужно написать две команды `onMouseOver`. Попробуем это реализовать. Эти две команды не разделяются. Мы хотим, чтобы они произошли одновременно, поэтому не будем разделять команды точкой с запятой, так как точка с запятой означает конец оператора.

Новое правило: ставьте запятую, чтобы отделить друг от друга несколько событий JavaScript.

А что насчет кавычек? Запомните, в кавычки помещают отдельные элементы вроде текста. Раз нам нужно, чтобы обе команды выполнились одновременно, как одна, то ставим кавычки в самом начале первой и в самом конце второй. Таким образом мы показываем браузеру, что все это одно событие. Понятно?

Однако нам еще могут понадобиться одинарные кавычки...

Вот что мы получим:

```
<A HREF="http://www.INTUIT.ru"
onMouseOver="document.bgColor='pink',
onMouseOver=window.status='Посетите INTUIT.RU';
return true">Щелкните здесь</A>
```

Таких обработчиков событий великое множество, и все они прекрасно работают. В следующем уроке мы поговорим еще о нескольких и научимся их комбинировать.

Вы заметили, что мы начинаем понемногу добираться до сути дела? Не

забывайте, что JavaScript очень логичный язык. Позднее мы проведем специальный урок, посвященный иерархии элементов.

Задание

В этом задании предлагается воспользоваться новым методом, `alert()` (предупредить). Он выводит небольшое диалоговое окно с текстом и кнопкой OK. Попробуйте сделать так, чтобы окно предупреждения появлялось при наведении курсора на ссылку. Вот формат команды:

```
alert('выводимый в окне текст')
```

Продумайте решение, решите, что должно произойти сначала, что потом. На самом деле все это довольно просто.

Возможное решение

```
<A HREF="http://www.INTUIT.ru"
onMouseOver="alert('Щелкните OK, чтобы закрыть окно');"
>Щелкните здесь</A>
```

Обратите внимание, схема остается фактически та же. Задание состояло в том, чтобы команда `onMouseOver` вызывала предупреждение. Значит, нужно оставить `onMouseOver`, добавить метод `alert()`, изменить текст, не забывая про двойные и одинарные кавычки, и все готово.

Еще несколько обработчиков событий

Описание основных событий и методы их обработки.

Концепция

В предыдущем уроке были рассмотрены некоторые понятия обработки событий.

Давайте рассмотрим теперь действие еще нескольких обработчиков событий. Все они работают по одной схеме. Поэтому, зная формат события и представляя логику его работы, можно легко поместить его на Web-странице.

Сценарии и результаты их работы

Команда onClick

Мы знаем, что команда `onMouseOver` запускает событие, если навести курсор на ссылку. Аналогично, щелкнув по ссылке, можно с таким же успехом запустить событие с помощью команды `onClick`.

Чтобы продемонстрировать действие команды, воспользуемся методом `alert`. Этот метод использовался в задании к прошлому уроку. Вот еще раз его схема:

```
alert('текст, который появится в окне')
```

Таким образом, применяя тот же подход, что и для `onMouseOver`, получаем код:

```
<A HREF="http://www.mail.ru"
onClick="alert('Посмотреть почту!');">
Щелкните здесь</A>
```

При щелчке на ссылке появится окно с сообщением "Посмотреть почту!"

Помните, что внутри одинарных кавычек нельзя употреблять слова с

апострофами ', иначе браузер поймет их как окончание текста, а это будет ошибка.

Команда onFocus

Это замечательный обработчик событий, который вызывает действие, когда пользователь "фокусируется" на элементе страницы. Он будет работать для элементов формы: флажков, текстовых полей, текстовых областей и др.

Вот пример:

```
<FORM>
<INPUT TYPE="text" SIZE="30"
onFocus="window.status='Текст в строке состояния';">
</FORM>
```

При использовании этого сценария выводится текстовое поле, при щелчке в котором в строке состояния выводится строка 'Текст в строке состояния'.

Команда onBlur

Если можно направить фокус на объект, значит, можно и "потерять фокус". Обработчик событий `onBlur` позволяет сообщить пользователю о том, что он изменил свой ответ. Этот обработчик не так часто используется, но приведем все-таки пример.

Создается текстовое поле с текстом. Надо изменить текст и затем щелкнуть мышью вне поля, имитируя переход фокуса к другому элементу.

Легко догадаться, как это делается, вот код:

```
<FORM>
<INPUT TYPE="text" SIZE="40"
VALUE="Впишите свое имя и щелкните вне текстового поля"
onBlur="alert('Вы изменили ответ — вы уверены, что он правильный?');
</FORM>
```

Команда onChange

Действие этой команды очень похоже на действие предыдущей, `onBlur`. Ее главная задача — проверка. Когда мы ближе познакомимся с формами, это станет понятнее.

Этот обработчик события проверяет, сделал ли пользователь то, о чем его просили (точнее, был ли изменён объект!). Пример очень похож на предыдущий, но действует по-другому.

```
<FORM>
<INPUT TYPE="text" SIZE="45"
VALUE="Измените текст и щелкните вне поля — затем
  проверьте строку состояния"
onChange="window.status='Текст был изменен'">
</FORM>
```

Т.е. при изменении текста в поле и последующем изменении фокуса в строке состояния выводится сообщение `'Текст был изменен'`.

Событие onSelect

Эта команда работает так же, как и три предыдущие, указывая, что в поле ввода формы произошли изменения. Отличие состоит в том, что данная команда реагирует, когда в поле ввода что-то было выделено.

Команда onSubmit

Это очень популярная команда. Она позволяет вызвать какое-либо действие, когда вы нажимаете кнопку Submit (отослать, отправить). Когда пользователь нажимает на эту кнопку, команда выведет на экран страницы текст: `"Спасибо, что вы нам написали"`.

Формат команды следующий:

```
<FORM>
<INPUT TYPE="submit"
onSubmit="parent.location='спасибо.html'";>
```

```
</FORM>
```

Здесь используются новые команды. Схема как будто знакомая, но какая-то странная.

`parent.location` — это стандартная схема ссылки на другую страницу. Можно подумать, что `parent` (источник) — это объект, а `location` (местонахождение) — метод. Неверно. В данном случае `parent` является свойством окна браузера, а `location` — объектом, который должен появиться в этом окне. То есть для ясности просто имейте в виду, что `parent.location=''` означает ссылку.

Команды onLoad и onUnload

Пример для этих команд здесь не приводится, так как каждой из них будет посвящен отдельный урок, но они являются обработчиками событий, поэтому надо по крайней мере их упомянуть.

Обе они помещаются внутрь команды `<BODY>` документа HTML. Они вызывают событие, когда страница открывается или закрывается, то есть когда пользователь уходит со страницы. Эти команды будут очень полезны при работе с функциями.

Задание

В этом задании предлагается создать форму, которая будет взаимодействовать с пользователем (для этого необходимо знать о командах формы).

Форма должна иметь три элемента: поле ввода с просьбой ввести имя; два поля для флажков с вопросом о том, что предпочитает пользователь — мороженое или шоколад; кнопку отправки данных (submit).

С каждым элементом должно произойти следующее.

- При вводе имени в строке состояния должны появиться слова: `Введите свое имя`.
- Два поля с флажками должны отослать в строку состояния слова:

`Вы выбрали...` и выбор пользователя.

- При нажатии на кнопку отправки должно появиться окно с благодарностью пользователю за участие в опросе.

Возможное решение

Это можно сделать с помощью следующего сценария:

```
<FORM action="">

Name: <INPUT TYPE="text" SIZE="30"
  onFocus="window.status='Введите свое имя';">
Что вы предпочитаете:
<INPUT TYPE="checkbox"
  onClick="window.status='Вы выбрали шоколад';">
      Шоколад
<INPUT TYPE="checkbox"
  onClick="window.status='Вы выбрали мороженое';">
      Мороженое
<INPUT TYPE="submit"
  onClick="alert('Спасибо за участие в опросе');">

</FORM>
```

Запросы пользователю и переменные

Работа с переменными. Запросы пользователю.

Концепция

В этом уроке рассматриваются две концепции. Одна из них используется, когда необходимо запросить у пользователя информацию. Вторая — создание переменных — будет постоянно применяться при работе с JavaScript.

Кстати, мы возвращаемся к созданию полноценных сценариев JavaScript, а не просто будем добавлять к HTML события, поэтому снова будет использоваться полный формат `<SCRIPT type="text/javascript">... </SCRIPT>`. Вот что мы собираемся сделать: Мы просим пользователя ввести имя, и с этим именем будет связана переменная. Когда переменная будет присвоена, мы сможем ввести ее в строку `document.write`, которая выведет имя пользователя на странице.

Этот урок немного длиннее, так как он включает краткий обзор того, что было изучено до сих пор.

Сценарий

```
<SCRIPT type="text/javascript">

/* Этот сценарий предназначен для получения информации
от пользователя и вывода ее на странице */

var user_name = prompt ("Введите свое имя в поле ниже","Здесь");
document.write("Привет, " + user_name + "!
  Добро пожаловать на мою страницу!");

</SCRIPT>
```

Текст в скобках должен располагаться на одной строке.

Результат работы сценария

Привет, OL! Добро пожаловать на мою страницу!

Разбор сценария

Что такое /* */?

Эти символы применяются для создания комментариев внутри сценария. В одном из предыдущих уроков мы узнали, что можно использовать двойную косую черту `//` для создания строки комментария. Двойную косую черту необходимо ставить в начале каждой новой строки комментария. Эти же команды позволяют создавать многострочные комментарии. Нужно поставить `/*` в начале, а `*/` в конце, и все, что окажется между этими знаками будет комментарием.

Создание переменной

Переменные имеют первостепенное значение в JavaScript. Необходимо знать, как их создавать. В двух словах: для вывода функции JavaScript задается имя, состоящее из одного слова. Вспомните, как мы помещали дату на страницу с помощью метода `getDate()`? В строке `document.write` мы написали метод `getDate()` целиком. Сделать это один раз было не так уж трудно. Но что, если нужно было бы написать это десять раз на одной и той же странице? Писать одно и то же порядком надоедает. Потому мы задаем переменную, которая должна представлять окончательный результат метода. Возьмем, к примеру, переменную `d`. Тогда нужно будет только один раз написать `getDate()` и назначить результату метода переменную `d`. И на протяжении всего оставшегося сценария будем просто писать `d` там, где нужно поставить дату. Вернемся к примеру.

Вот строка из скрипта, которая задает переменную:

```
var user_name = prompt ("Введите свое имя в поле ниже","Здесь")
```

Переменная была создана по следующей схеме.

- `var` (от variable, переменная) объявляет, что следующим словом будет имя переменной.
- `user_name` (имя_пользователя) — имя переменной. Произвольное. Оно не обязательно должно быть таким длинным. Можно было бы использовать при желании просто `N`. Но удобнее называть переменные так, чтобы легко было вспомнить, о чем идет речь.
- Помните, что регистр имеет значение для JavaScript, следовательно, если переменная обозначена `Dog`, то буква D каждый раз должна быть заглавной, иначе браузер посчитает их за два разных слова.
- Здесь нет никаких кавычек, просто ставьте одно слово за другим, как показано выше.
- Знак равенства = указывает на то, что переменная будет равна результату следующей команды.
- В данном случае переменная будет представлять результат, полученный с помощью окна запроса

Команда Prompt

В данном примере используется новая команда `prompt` (запрос). Этот метод выводит окно с сообщением и полем ввода. Вот формат запроса:

```
var variable_name = prompt("Текст окна","Текст в поле ввода")
```

Можно видеть, что `var` и присваиваемое имя переменной включены в формат. Иначе получился бы запрос, но ничего нельзя было сделать с данными, которые вводит пользователь.

На всякий случай ...

- Чтобы строка ввода оставалась пустой, ничего не пишите между второй парой кавычек.
- Если не указать в скобках второй пары кавычек, в строке ввода появится слово "undefined".
- Если в строке ввода что-то написано и пользователь выберет OK,

ничего не меняя, то в качестве вывода на странице появится то, что написано в строке ввода.

- Если в строке ввода ничего нет и пользователь выберет OK, ничего не вводя, то на странице появится слово `null`.

Вернемся к разбору

Теперь, зная все составляющие блоки, вернемся снова к сценарию:

```
var user_name = prompt ("Введите свое имя в поле ниже","Здесь");
document.write("Привет, " + user_name +
"!Добро пожаловать на мою страницу!");
```

Вот весь процесс.

1. Имя переменной user_name присвоено результату запроса.
2. prompt просит пользователя написать свое имя в поле ввода.
3. В поле ввода записано: "Здесь."
4. Точка с запятой в конце строки.
5. `document.write` вызывает текст `"Привет, "`.
6. Знак плюс `+` указывает, что все элементы идут друг за другом.
7. `user_name` содержит результат запроса. Никаких кавычек — нам на странице нужен результат запроса, а не эти слова.
8. Еще плюс.
9. "!Добро пожаловать на мою страницу!" завершает текст.
10. Точка с запятой.

Пожалуйста, постарайтесь хорошо понять концепцию переменных. В JavaScript они используются постоянно. Без них не обойтись.

Задание

...повторение пройденного.

Мы изучили шесть уроков. Пора приниматься за более сложные сценарии. Мы имеем уже достаточно знаний для создания сложных эффектов с помощью команд JavaScript.

В качестве сегодняшнего задания необходимо выполнить анализ показанного ниже сценария, объяснив, как работает каждая его часть. Может показаться удивительным, но любой сценарий без труда раскладывается на легко понятные части, как только станет понятно, что происходит в сценарии.

Задание 6. Самостоятельный разбор сценария

Вот сценарий. Разберите его на составляющие элементы, как мы делали в ходе урока.

```
<SCRIPT type="text/javascript">

var name =
  prompt("Пожалуйста, напишите свое имя","")

var d = new Date();

var y =  d.getFullYear();
var m = d.getMonth() + 1;
var d = d.getDate();

var t =  m + '/' + d + '/' + y + ' ';

document.write("<TITLE>")
document.write
("Привет "+name+ ". Сегодня "
+t+ ". Спасибо, что зашли.");
document.write("</TITLE>")

</SCRIPT>
```

Разбор задания

- Начинаем со строки `<SCRIPT LANGUAGE="javascript">`.
- Создается переменная `name`. Имя пользователь вводит по запросу `prompt`.
- Нам нужна дата, поэтому для даты создается переменная `d`.
- Мы знаем, что год, месяц и число определяются с помощью

`d.getЧто-либо()`, поэтому создаем переменные для каждого из этих элементов: `y`, `m` и `d`.

- Видите `+ 1` после переменной `m`? Так мы получаем правильный месяц.
- Для полностью записанной даты, состоящей из числа, месяца и года, также задается переменная с именем `t`.
- Команда `TITLE` создается с помощью трех команд `document.write`. Таким образом легче всего отделить команды HTML и код JavaScript.
- Первая команда из трех начинает команду `<TITLE>`.
- Второй оператор создает строку текста. Обратите внимание, что выводимый текст помещен в двойные кавычки и отделен от команд, отвечающих за дату, знаками плюс. Не забудьте о пробелах в тексте.
- Третья команда `document.write` завершает команду `<TITLE>`.
- И все завершает `</SCRIPT>`.
- Важно также помнить, что этот сценарий должен идти в документе HTML перед командой `BODY`, так как `TITLE` располагается в этом месте.

Концепция свойств

Объектная модель документа. Основные объекты и их свойства.

Концепция

Иерархия объектов JavaScript играет настолько важную роль, что мы посвятим ей отдельный урок, но мы уже и так начинаем понемногу ее осваивать.

Нам известно, что существуют объекты, например, `document`, и методы, например, `write`, которые воздействуют на объекты. В Уроке 6 мы немного имели дело с созданием объектов или созданием переменных. Теперь рассмотрим концепцию свойств. Мы уже слегка касались этой темы. Свойства представляют собой часть или качество объекта. В Уроке 4 мы работали со свойством документа `bgColor`.

Было бы трудно описать все возможные свойства в одном уроке. Поэтому здесь будут рассмотрены только наиболее популярные из них и указано, какую пользу они могут принести.

Сценарий

Ниже вы увидите несколько сценариев, но все они составлены по одной схеме: для каждой команды `объект.свойство` (`object.property`) создается переменная, затем переменные помещаются в команду `document.write()` для вывода.

Свойства объекта "navigator" (браузер)

```
<SCRIPT LANGUAGE="javascript">
var an = navigator.appName;
var av = navigator.appVersion;
var acn = navigator.appCodeName;
var ua = navigator.userAgent;
document.write("Вы пользуетесь <B>" +an+ "</B>,
версия " +av+ ".
```

```
<BR>Кодовое название " +acn+ ", заголовок "
+ua+ "." );
</SCRIPT>
```

Приведенный выше текст в скобках должен располагаться на одной строке.

Свойства объекта "document"

```
<SCRIPT LANGUAGE="javascript">
var bgc = document.bgColor;
var fgc = document.fgColor;
var lc = document.linkColor;
var al = document.alinkColor;
var vlc = document.vlinkColor;
var url = document.location;
var ref = document.referrer;
var t = document.title;
var lm = document.lastModified;
document.write("Цвет фона этой страницы <B>"
+bgc+ "</B>.")
document.write("<BR>Цвет текста этой страницы <B>" +fgc+ "</B>.")
document.write("<BR>Цвет ссылок этой страницы <B>" +lc+ "</B>.")
document.write("<BR>Цвет активной ссылки этой страницы
<B>" +al+ "</B>.")
document.write("<BR>Цвет посещенной ссылки этой страницы
<B>" +vlc+ "</B>.")
document.write("<BR>URL этой страницы <B>" +url+ "</B>.")
document.write("<BR>До этого вы были на странице <B>" +
ref+ "</B>.")
document.write("<BR>Заголовок этой страницы (TITLE) <B>" +t+ "</B>.")
document.write("<BR>Последние изменения в документ внесены: <B>"
+lm+ "</B>.")
</SCRIPT>
```

Свойства объекта "history"

```
<SCRIPT LANGUAGE="javascript">
```

```
var h = history.length;
document.write("До этой страницы вы посетили" +h+ " страниц.")
</SCRIPT>
```

Два свойства объекта "location" (адрес)

```
<SCRIPT LANGUAGE="javascript">
var hst = location.host
document.write("Страница находится на <B>" + hst +
"</B>." )
</SCRIPT>

<SCRIPT LANGUAGE="javascript">
var hstn = location.hostname
document.write("Страница находится на <B>" + hstn +
"</B>." )
</SCRIPT>
```

Результат работы сценариев (возможный)

Некоторые сведения о вашем компьютере.

Вы пользуетесь Opera, версия 9.01 (Windows NT 5.1; U; ru).
Кодовое название Mozilla, заголовок Opera/9.01 (Windows NT 5.1; U; ru).

Цвет фона этой страницы #334775.
Цвет текста этой страницы #334775.
Цвет ссылок этой страницы #fbbcac.
Цвет активной ссылки этой страницы #d0550b.
Цвет посещенной ссылки этой страницы #fbbcac.
URL этой страницы http://jsp.newmail.ru/les7.htm.

До этого вы были на странице .

Заголовок этой страницы Урок 7. Концепция свойств.
Последние изменения внесены: Fri, 27 Jun 2003 00:09:36 GMT.

До этой страницы вы посетили 1 страниц.

Страница находится на www.mail.ru.

Страница находится на www.mail.ru.

Разбор сценария

Поговорим о каждой группе отдельно...

Почему в некоторых местах шрифт жирный?

Да в общем-то так, ради забавы. Найдите в сценарии элементы, которые выводятся жирным шрифтом. Как можно видеть, были просто добавлены команды `<B>` и `</B>` по обеим сторонам от имени переменной — внутри двойных кавычек. Раз это оператор `document.write`, то в текст можно вписать любые команды HTML, изменяющие текст. Только следите за тем, чтобы команды HTML находились внутри двойных кавычек, чтобы они воспринимались как текст, иначе браузер посчитает их частью скрипта — это было бы ошибкой.

Вернемся к нашим свойствам...

Свойства объекта navigator

```
<SCRIPT LANGUAGE="javascript">
var an = navigator.appName;
var av = navigator.appVersion;
var acn = navigator.appCodeName;
var ua = navigator.userAgent;

document.write("Вы пользуетесь <B>" +an+ "</B>,
версия " +av+ ".<BR>Кодовое название " +acn+ ",
заголовок " +ua+ "." );
</SCRIPT>
```

Еще раз напоминаем, что текст в скобках должен быть весь на одной строке.

Люди любят эти свойства. Объект `navigator` имеет четыре свойства. Обратите внимание на заглавные буквы!

- `appName` сообщает название браузера, например, Netscape или Explorer.
- `appVersion` сообщает версию браузера и платформу, на которой он работает.
- `appCodeName` сообщает кодовое имя, данное браузеру, например, Netscape называет свой браузер Mozilla.
- `userAgent` сообщает версию используемого браузере.

Иногда важно знать версию браузера. Чуть позже мы изучим команды `if` (если). Зная браузер пользователя и его версию, можно дать команду: "Если браузер такой-то, сделать то-то."

Свойства объекта document

```
<SCRIPT LANGUAGE="javascript">
var bgc = document.bgColor;
var fgc = document.fgColor;
var lc = document.linkColor;
var al = document.alinkColor;
var vlc = document.vlinkColor;
var url = document.location;
var ref = document.referrer;
var t = document.title;
var lm = document.lastModified;
document.write("Цвет фона этой страницы <B>"
+bgc+ "</B>.")
document.write("<BR>Цвет текста этой страницы <B>" +fgc+ "</B>.")
document.write("<BR>Цвет ссылок этой страницы <B>" +lc+ "</B>.")
document.write("<BR>Цвет активной ссылки этой страницы
<B>" +al+ "</B>.")
document.write("<BR>Цвет посещенной ссылки этой страницы
<B>" +vlc+ "</B>.")
document.write("<BR>URL этой страницы <B>" +url+ "</B>.")
document.write("<BR>До этого вы были на странице <B>" +
ref+ "</B>.")
```

```
document.write("<BR>Заголовок этой страницы <B>" +t+ "</B>.")
document.write("<BR>Последние изменения в документ внесены: <B>"
+lm+ "</B>.")
</SCRIPT>
```

Снова напоминаем, что текст выше в скобках должен целиком располагаться в одной строке.

Свойства документа HTML очень популярны в JavaScript. Здесь перечислены девять. На самом деле их тринадцать, но остальные четыре нам пока не нужны. Они перечислены ниже.

- `bgColor` возвращает шестнадцатеричный код цвет фона.
- `fgColor` возвращает шестнадцатеричный код цвета текста.
- `linkColor` возвращает шестнадцатеричный код цвета ссылки.
- `alinkColor` возвращает шестнадцатеричный код цвета активной ссылки.
- `vlinkColor` возвращает шестнадцатеричный код цвета посещенной ссылки.
- `location` возвращает URL страницы.
- `referrer` сообщает, с какой страницы пришел пользователь. Если информация недоступна, то возвращается пустое место.
- `title` возвращает заголовок документа HTML, т .е. текст между командами `TITLE`.
- `lastModified` сообщает дату, когда были внесены последние изменения в страницу (на самом деле дату, когда страница была загружена на сервер или сохранена последний раз на жестком диске).
- `cookie` (не показано) возвращает текстовый файл cookie.
- `anchors` (не показано) возвращает количество анкеров HREF на странице.
- `forms` (не показано) возвращает массив (список) объектов формы на странице.
- `links` (не показано) возвращает количество всех отдельных ссылок.

В данном случае также с помощью команды `if` можно сказать: "Если время больше 6 вечера, пусть текст будет белый, а фон черный. Если еще

нет 6 вечера, то пусть фон будет голубой, а текст зеленый". Существует множество способов использовать свойства документа.

Свойства объекта history

```
<SCRIPT LANGUAGE="javascript">
var h = history.length;
document.write("До этого вы посетили " +h+ " страниц.")
</SCRIPT>
```

Это очень популярный объект. Многие читатели хотят иметь возможность переместиться на одну или несколько страниц вперед или назад. Они пытаются воспроизвести кнопки "Вперед" и "Назад" на панели браузера. Объект `history` позволяет это сделать.

Объектом является журнал посещений `history`. Это список страниц, которые посетил браузер во время работы. Список истории позволяет реализовать кнопку "Назад" и просмотреть еще раз любую страницу.

Свойством является `length` (протяженность). Оно также популярно. Позже вы узнаете, как можно использовать его с другими командами. Оно позволяет определить в сценарии количество страниц в папке "history".

Существует также метод `go()` (пойти), который позволяет передвигаться по `history.length` с указанным шагом.

Два свойства объекта location

```
<SCRIPT LANGUAGE="javascript">
var hst = location.host
document.write("Страница находится на <B>" + hst + "</B>.")
</SCRIPT>

<SCRIPT LANGUAGE="javascript">
var hstn = location.hostname
document.write("Страница находится на <B>" + hstn + "</B>.")
</SCRIPT>
```

Здесь объектом является `location`. Это URL на языке JavaScript, адрес страницы. Выше представлены два свойства объекта `location`: `host`, и `hostname`. Команды равноценны, так как выполняют одну и ту же задачу — сообщают URL в текстовом формате или адрес IP, в зависимости от сервера. Но... `location.host` сообщает URL плюс "порт", с которым соединен пользователь. `location.hostname` сообщает только URL.

Если вы получаете одинаковый результат от обеих команд, значит, ваш сервер не соединил вас со специальным портом. Говоря техническим языком, свойство "порта" — `null`.

Кстати, эти две команды не работают, если просматривать страницу с жесткого диска. Результат может быть только в том случае, если она размещается на сервере, так как сценарию требуется URL для анализа.

Существует множество других свойств, с которыми вы встретитесь во время уроков. Здесь даны общие представление о свойствах — как они используются и что делают наиболее часто используемые.

Задание

Задание будет следующее: используя одну из вышеприведенных команд типа `объект.свойство`, напишите сценарий JavaScript, который создает ссылку на страницу документа HTML на каком-либо сервере. Например, если вы находитесь на ссылка: www.you.ru - http://www.you.ru, сценарий JavaScript создаст ссылку на ссылка: www.you.ru/index.html - http://www.you.ru/index.html.

Также, какое бы свойство ни использовалось, присвойте ему имя переменной.

Возможное решение

Это можно было сделать по-разному, например, так:

```
<script LANGUAGE="javascript">
```

```
  var dr = location.host;
  document.write("<a HREF=http://" +dr+ "/index.htm>
    Щелкните, чтобы попасть куда-нибудь</a>")

  </SCRIPT>
```

Имейте в виду, что страница должна находиться на сервере, на жестком диске сценарий не работает, так как там нет никакого `location.host`.

Иерархия объектов

Структура объектов JavaScript, порядок их подчинения.

Концепция

Сделаем паузу и посмотрим, что мы уже знаем. В JavaScript есть объекты, похожие на существительные или предметы. У объектов есть свойства, которые описывают их, как прилагательные описывают существительное. Мы ссылаемся на свойства с помощью схемы объект.свойство.

Еще у объектов есть методы, или действия, которые можно выполнить с объектом. Все методы имеют скобки и используются по схеме `объект.метод()`. У разных объектов имеются разные свойства и методы.

Теперь мы познакомимся с иерархией объектов JavaScript. Как только вы ее поймете, считайте, что вы освоили JavaScript!

Что имеется в виду

- Window
  - Parent
  - Self
    - Location
      - Href
    - Document
      - Image
        - Src
      - Form
        - Text
        - Submit
        - Checkbox
  - Top
  - Frames

Результат действия иерархии

Все ссылки начинаются с наивысшего объекта, `window` (окно браузера), и идут по нисходящей. Окна и рамки (frames) принадлежат объекту `window`. На них не нужно ссылаться, если только их не больше одного. `Top`, `self`, `parent` и `frames` — "встроенные" имена для окон. Не придавайте им большого значения, просто знайте, что они существуют.

Вот несколько примеров. Обратите внимание на иерархию.

```
document.mypic.src = "pic1.gif"
```

в самом начале `window` не требуется. Предполагается, что это все и так находится внутри окна. Команда `document.mypic.src` указывает на изображение с именем `mypic`, и изменяет его содержимое на `"pic1.gif"`. В данном случае `document` (документ) — это страница на которой находится элемент, `mypic` — имя элемента, а `SRC` — источник элемента (`"pic1.gif"`).

```
document.write(location.href)
```

`write()` — это метод объекта `document`. `location.href` содержит полный URL окна. Обратите внимание, что `location` и `document` находятся на одном уровне. Это значит, что вы получаете адрес документа того же уровня.

Разбор иерархии объектов

- Window
  - Parent
  - Self
    - Location
      - Href
    - Document
      - Image
        - Src
      - Form
        - Text

        - Submit
        - Checkbox
    - Top
    - Frames

- Самая большая путаница в том, что некоторые объекты также являются и свойствами.
- `window` — только объект.
- `document` является свойством окна, но в свою очередь и объектом.
- `form` — это свойство документа, но также и объект со своими свойствами!
- `value` (значение) и `SRC` (источник) — только свойства!
- Здесь представлены не все объекты и свойства. Однако этого достаточно, чтобы понять концепцию в целом... Все ссылки начинаются сверху от `window` и идут по нисходящей. То есть, нельзя написать `document.mytext.myform` или `mypic.src.document`. Это неправильный порядок, следует писать слева направо от более общего к более конкретному.
- Важное замечание: чтобы показать содержимое поля формы, необходимо использовать свойство `value` (значение), например, `document.myform.mytext.value`! Если написать просто `document.myform.mytext`, то будет получена информация о поле формы, но не о его содержании!

Считайте `value` ("значение") некоторым показателем того, что нечто имеется или отсутствует в определенное время. Поле с флажком может иметь значение "on" или "off", в зависимости от того, задан он или нет. Текстовое поле может иметь значение "hidden" (скрытое), если вы не хотите, чтобы пользователь его видел. Текстовое поле, как указано выше, может содержать какую-то запись. Она будет значением этого поля.

Задание

Напишите полные ссылки, начиная с `window`, даже если это не требуется. Обратите внимание, что изображение имеет имя `mypic`, а

форма — `myform`.

1. Ссылка на всю форму:
2. Ссылка на содержимое поля `lname`:
3. Ссылка на содержимое поля `fname`:
4. Замените изображение на `"marigold.gif"`:

Ответы

1. `window.document.myform`
2. `window.document.myform.lname.value`
3. `window.document.myform.fname.value`
4. `window.document.mypic.src=marigold.gif`

Создание функций

Функции в JavaScript. Их создание и использование.

Концепция

При создании переменной результату команды или события JavaScript присваивается имя. При создании функции делают почти то же самое, только имя присваивается целой серии команд. Множество команд JavaScript объединяются в одну.

Сценарий

Сценарий состоит фактически из двух частей: собственно функции и команды `onLoad`, которая запускает функцию в работу.

Вот обе части:

```
<SCRIPT LANGUAGE="javascript">

<!-- Скрыть от браузеров, не поддерживающих JavaScript

function dateinbar()
{
var d = new Date();
var y = d.getFullYear();
var da = d.getDate();
var m = d.getMonth() + 1;
var t = da + '/' + m + '/' + y;

defaultStatus = "Вы прибыли на страницу " + t + ".";
}
// не скрывать -->

</SCRIPT>
```

...и команда `onLoad` в `<BODY>`:

```
<BODY BGCOLOR="xxxxxx" onLoad="dateinbar()">
```

Результат работы сценария

Результат работы сценария будет выводиться внизу в строке состояния.

Почти таким же сценарием мы пользовались для получения даты на прошлых уроках, поэтому он должен быть понятен. Здесь нас интересует реализация сценария.

Разбор сценария

Здесь мы видим новые элементы `<!--` и `-->`. Возможно, они покажутся знакомыми. Эти знаки используются для комментариев в теле HTML:

```
<!-- Это текст комментария, не видимый на странице -->
```

Они присутствуют здесь, так как, хотите верьте, хотите нет, но все еще существуют браузеры, не поддерживающие JavaScript. Если браузер не поддерживает JavaScript, он воспринимает его как текст, который нужно вывести на странице. Выглядит это плохо. Но если воспользоваться этими командами комментария, то браузер успешно проигнорирует незнакомый текст и покажет страницу.

Однако соблюдайте несколько правил.

- Команды комментария должны находиться между `<SCRIPT>` и `</SCRIPT>`. Если поставить их снаружи, то браузер посчитает комментарием весь сценарий JavaScript и ничего не будет выполнено.
- Текст после команды `<!--` должен находиться на одной строке.
- Перед заключительной командой `-->` должна стоять двойная косая черта `//`, иначе JavaScript примет ее за часть сценария, что приведет к ошибке. Можно также добавить некоторый текст, так как он будет считаться комментарием.
- Совсем не обязательно писать текст к этим командам. Текст присутствует для того, чтобы проще было понять их назначение.

Пользуйтесь приведенной выше схемой, и проблем не будет.

Вернемся к разбору

Здесь происходит две вещи. Сначала в первой части сценария создается функция. Потом команда, находящаяся внутри оператора HTML `<BODY>`, запускает работу функции. Давайте сначала разберем функцию.

```
function dateinbar()
{
var d = new Date();
var y = d.getFullYear();
var da = d.getDate();
var m = d.getMonth() + 1;
var t = da + '/' + m + '/' + y;
defaultStatus = "Вы прибыли на страницу " + t + ".";
}
```

Схема довольно понятная. Вы пишете `function` и задаете любое имя, точно так же, как мы делали с переменными.

Но обратите внимание, что после имени функции стоят круглые скобки, как после команды метода. То есть, при создании функции фактически создается новый метод для выполнения некоторой задачи.

Так же, как и имена переменных, имена функций могут быть любой длины, не содержать пробелов и не совпадать с другим словом, уже используемым в языке JavaScript. В данном случае мы выбрали имя `dateinbar()` (дата в строке состояния), потому что это функция и делает — помещает дату в строку состояния.

Очень важно!

При создании функции команды, которые составляют функцию, должны быть заключены в фигурные скобки `{}`. Видите, они стоят сразу после имени функции и в самом конце?

Текст внутри фигурных скобок должен быть понятен. Такой же сценарий мы использовали пару уроков назад.

- создается переменная для года;
- еще одна для числа;
- еще одна для месяца;
- затем четвертая для даты целиком.

Последняя команда новая:

```
defaultStatus = "Вы прибыли на страницу " + t + ".";
```

`defaultStatus` (строка состояния по умолчанию) — свойство объекта `window`. Его назначение — поместить текст в строку состояния внизу окна браузера.

Но почему не написать просто `window.status`?

Хороший вопрос. Этого нельзя сделать, потому что такой формат используется для события, например, с командой `onClick`. Раз строка состояния не находится внутри команды HTML, то используем вместо этого `defaultStatus`. Есть только одна строка состояния — она же и по умолчанию.

Команда "onLoad="

Мы задействуем новый обработчик событий. Команда `onLoad` ("при загрузке") (обратите внимание на заглавные буквы) говорит браузеру, что, загружая страницу, он должен выполнить предписанное далее. В данном случае следует функция `dateinbar()`.

Эта команда почти всегда располагается в строке `<BODY>` документа HTML. И почти всегда за ней следует функция, но это необязательно. Можно с таким же успехом поместить туда и команду `объект.метод`.

Расположение элементов

Это имеет не последнее значение. Вы знаете, что команда `onLoad` идет

в строку BODY. Сценарий с функцией должен находиться в документе HTML между командами <HEAD> и </HEAD>. Хотя на самом деле его можно поместить где угодно, но если вы расположите его после команды onLoad, он заработает только после того, как загрузится вся страница. Если сценарий располагается перед командой onLoad, то он загружается в память компьютера раньше, и когда onLoad вызовет его, он будет готов к работе.

Практически любой набор команд JavaScript можно записать в виде функции.

Задание

Это задание достаточно сложное. Создайте функцию, которая создает два запроса (prompt). (Подсказка: один следует за другим с новой строки.) Первый предлагает пользователю ввести свое имя, второй — фамилию. Затем та же функция должна вызвать окно сообщения (alert) с текстом:

> Привет, "имя фамилия", добро пожаловать на "адрес страницы",
> мою замечательную страницу!

Не забудьте создать переменную для адреса страницы.

Можно немного усложнить, делая сценарий так, что слова "мою замечательную страницу" будут вставлены в текст команды alert не просто, а с помощью дополнительной переменной.

Возможное решение

Это задание требует известной сообразительности. Нужны две команды prompt, одна сразу за другой. Затем какая-то переменная, которая напечатала бы название страницы (например, "Моя замечательная страница"). Мы сделали это, поместив текст в заголовок и создав переменную для команды document.title. Потом потребуется переменная для document.location.

Далее они используются в команде alert. Все это содержится в

функции с именем `hello()`, а функция должна запускаться командой `onLoad` в строке `BODY`.

Можно было бы также создать переменную для строки текста, но, так как она записывается только один раз, это было бы ни к чему. Вот сценарий, который все это делает:

```
<SCRIPT LANGUAGE="javascript">

function hello()
{
var first = prompt("Введите свое имя", "Имя")
var second = prompt("Введите свою фамилию", "Фамилия")
var page = document.location;
var ttl = document.title;
alert("Привет, " +first+ " " +second+ ".
  Добро пожаловать на " +ttl+ " по адресу: " +page+".")
}

</SCRIPT>
```

А вот команда `BODY` в документе:

```
<body bgcolor="ffffcc" onLoad="hello()">
```

Команды последействия: onUnLoad и onMouseOut

Использование событий onUnLoad и onMouseOut

Концепция

Это два последних обработчика событий, которые необходимо иметь в своем арсенале: `onMouseOut` и `onUnload` (обратите внимание на заглавные буквы). Они рассматриваются в одном уроке, потому что начинают действовать после того, как что-то сделано.

`onMouseOver` вызывает некое событие, если навести мышь, к примеру, на ссылку. В противоположность ей `onMouseOut` начинает действовать, если курсор увести со ссылки. Мы также знаем, что команда `onLoad` запускает сценарий, когда страница загружается. Команда `onUnload` действует, когда пользователь уходит со страницы.

Сценарий

Следующий код использует события при перемещении указателя мыши:

```
<A HREF="index.htm" onMouseOver="window.status='Эй! Убирайся с меня return true"
onMouseOut="window.status='Так-то лучше, спасибо'; return true">
Наведите курсор на эту ссылку и сместите в сторону</A>
```

Использование команды `onUnload` при уходе со страницы:

```
<BODY onUnload="alert('Уже уходите?')">
```

Результат работы сценария

При размещении сценария на странице выводится текстовая ссылка "Наведите курсор на эту ссылку и сместите в сторону". Если навести курсор на ссылку и сместить в сторону несколько раз, то в строке состояния можно видеть изменяющиеся сообщения. Это первый результат. При нажатии на ссылку можно увидеть второй.

Разбор сценария

Эффекты с мышью, как легко видеть, создаются с помощью команд `onMouseOver` и `onMouseOut`.

Обратите внимание, что эти две команды никак не связаны между собой. Вам не нужно, чтобы эти события происходили одновременно. Помните, несколько уроков назад мы разделяли два обработчика событий запятой, чтобы они происходили одновременно? Здесь не тот случай. Требуется, чтобы одно событие происходило, когда курсор мыши указывает на ссылку, а другое — когда курсор мыши смещается со ссылки. Поэтому нужно писать их как две совершенно независимые команды, каждая из которых содержит свою команду `return true`.

Сообщение при уходе со страницы создается с помощью команды `onUnload="alert('Уже уходите?')"`, которая добавлена в строку `BODY` документа HTML. Обратите внимание на двойные и одинарные кавычки. Внутри двойных — одинарные. Вторая двойная кавычка означает для браузера конец команды.

Задание

В этом задании необходимо использовать функции `onUnload`, `onMouseOver`, и `onMouseOut`. Сделайте следующее:

- Создайте страницу с гипертекстовой ссылкой.
- Когда курсор указывает на ссылку, в строке состояния должны появляться слова: "Привет, пользователь 'название браузера'!".
- Когда курсор уходит со ссылки, в строке состояния должен появляться текст: "Не скучаете у нас на 'URL страницы'?"
- Если щелкнуть на ссылке, должно появиться окно со словами: "Уже уходите? Сейчас всего только 'текущее время'";
- Время должно определяться с помощью функции.
- Не пользуйтесь командой `onClick`, чтобы вывести окно сообщения, возьмите команду `onUnload`.

Возможное решение

Вот сценарий, который реализует задание:

```
<HEAD>

<SCRIPT LANGUAGE="javascript">

  <!-- Скрываем от браузеров, которые не
    поддерживают Javascript

  function goodbyedate()
  {
  var d = new Date();

  var h = d.getHours();
  var m = d.getMinutes();

  var t = h + ':' + m + ' ';

  alert
    (" Уже уходите?  Сейчас всего только " + t + ".");
  }

  // конец сокрытия -->

  </SCRIPT>
</HEAD>

<body bgcolor="FFFFCC" onUnload="goodbyedate();">

<A HREF="assign_10.html"
  onMouseOver="window.status=
   'Привет, пользователь ' +navigator.appName+
     ' — щелкни здесь!';
  return true" onMouseOut="window.status=
   'Не скучаете у нас на ' +document.location+
     '.';
  return true">Проведите курсор мыши над этой ссылкой</A>

</BODY>
```

Функция, определенная в заголовке (между командами `<head>`) задает время. К этой функции обращается команда `onUnload` в строке `<body>` документа HTML. Переменная времени используется в команде `alert`.

Команды `onMouseOver` и `onMouseOut` построены по той же схеме, что и в уроке, кроме переменных `navigator.appName` в `onMouseOver` и `document.location` в `onMouseOut`. Не запутайтесь с двойными и одинарными кавычками. Каждая из команд `window.status` помещена в двойные кавычки. Отрезки текста стоят в одинарных кавычках. Внимательно закрывайте все кавычки и скобки, и тогда у вас не будет больших проблем.

Открываем новые окна

Методы создания и работы с новыми окнами.

Концепция

Это первый из двух уроков, посвященных открытию новых окон. Первый урок имеет дело с командами Javascript, которые используются для открытия нового окна. В новом окне будет выводиться другой документ HTML.

Второй урок также рассматривает создание нового окна, но с помощью функции — так, чтобы оба окна содержали одну и ту же страницу. Не нужно будет использовать две страницы HTML. Все можно будет сделать с помощью одной страницы HTML и сценария JavaScript.

Начнем с основ.

Сценарий

```
<SCRIPT type="text/javascript" >

window.open('opened.html', 'joe', config='height=300,width=300')

</SCRIPT>
```

Результат работы сценария

Обратите внимание! Представленный здесь сценарий будет только открывать окно. В этом окне выводится документ opened.html. Содержание документа будет рассмотрено позже.

Разбор сценария

Расположение на странице

Начнем с расположения сценария на странице. До сих пор всегда

говорилось, что лучше помещать скрипты выше в документе, чтобы они быстрее загружались в память компьютера и начинали работать без задержки. Когда речь идет о функции, сценарий помещается между командами <HEAD>. Здесь будет сделано другое предложение.

Если вы собираетесь открывать новое окно, поместите команды, которые это делают, ближе к концу документа HTML. Проще говоря, пусть они идут в последнюю очередь. Причина простая: сначала загрузится страница, а потом откроется окно. Если команды стоят в начале, то окно появится прежде, чем пользователь увидит страницу. Скорее всего он закроет новое окно, прежде чем его можно будет использовать.

Но это только частное мнение. На самом деле можно ставить эти команды в любом месте документа. Но все-таки, думается, что чем позже откроется новое окно, тем это лучше для посетителей страницы.

window.open

Нельзя сказать яснее, что `window` (окно) — объект, а `open` (открыть) — метод, который на него воздействует. Теперь перейдем к конфигурации окна.

Конфигурация нового окна

Это все, что находится в экземпляре команды (в круглых скобках, помните?) Вот формат, которому необходимо следовать:

```
('URL документа в окне', 'Название нового окна',
  config='параметры нового окна')
```

В рассматриваемом случае мы имеем:

```
('opened.html', 'joe', config='height=300,width=300')
```

- opened.html — это URL страницы, которая появится в новом окне. Если страница располагается на другом сервере, то необходимо добавить http:// и так далее.

- `joe` — имя нового окна. Далее это будет важно.
- `config=` указывает, что следующие команды относятся к конфигурации нового окна.

Команды config

Приведенные выше команды `config` открывают новое окно размером `300` на `300` пикселей.

Кстати, всегда делайте окно немножко больше, чем вам нужно. Может быть, у кого-то другое разрешение экрана, и ваши размеры окна ему не подойдут.

Обратите внимание, что команды `height` (высота) и `width` (ширина) разделены только запятой без пробелов, а значения поставлены в одинарные кавычки, так как эти два элемента являются подкомандами `config` и должны выполняться совместно. Пробел для браузера означает конец команды. Ошибка.

Для команды `config` существует множество подкоманд. Про высоту (`height`) и ширину (`width`) мы уже знаем, они определяются в пикселях. Остальные подкоманды употребляются со словами "yes" или "no" в зависимости от того, нужны ли в новом окне эти элементы. (Можно ставить "1" вместо "yes" и "0" вместо "no", но это не обязательно.)

Помните, никаких пробелов между подкомандами и используйте одинарные кавычки. Пробел равносилен ошибке.

- `toolbar=` отвечает за наличие панели инструментов во вновь открытом окне. Панель инструментов в верхней части окна браузера содержит кнопки НАЗАД, ВПЕРЕД, СТОП и т.д. ()
- `menubar=` отвечает за наличие строки меню с элементами ФАЙЛ, ПРАВКА, ВИД и т.д.
- `scrollbars=` отвечает за наличие полосы прокрутки.
- `resizable=` указывает, может ли пользователь изменить размер окна по своему желанию.
- `location=` отвечает за наличие адресной строки во вновь

открытом окне, в которой выводится URL страницы.

- `directories=` отвечает за наличие строки каталогов в новом окне, которая содержит закладки и т.п.
- `status=` отвечает за наличие строки состояния.

От строки с заголовком избавиться невозможно, хотите вы этого или нет.

Может быть, вы считаете, что все вышеперечисленное — свойства. Нет. Если вам проще их запомнить, считая свойствами, — отлично, считайте их чем угодно. Но в действительности они называются характеристиками или атрибутами. Они действуют как параметры события JavaScript. А в общем, назови хоть горшком, научись только пользоваться.

Тэги в новом окне

В открывающееся новое окно загружается документ HTML opened.html. Это обычная страница HTML, которая может содержать любые команды, в частности, с этой страницы можно управлять загрузкой других документов. Например, чтобы открыть главную страницу INTUIT в основном окне, надо поместить на ней следующий код:

```
<A HREF="http://www.intuit.ru" TARGET="main window"></A>
```

Основное окно всегда имеет по умолчанию имя "main". Поэтому в команду `HREF` документа HTML добавляется просто команда `TARGET="–"` с указанием main для окна, в которое должна загрузиться страница.

А если надо, чтобы страница загрузилась в новом окне? У этого окна, как было сказано выше, тоже есть имя, здесь оно названо `joe`. Необходимо написать просто команду ссылки `HREF` с указанием окна `joe`.

Можно открыть на самом деле несколько окон, добавляя несколько команд `window.open`. Надо только задать окнам различные имена. Можно создавать также ссылки между окнами, указывая необходимые имена окон.

Закрытие окна

Можно создать также в документе ссылку, которая будет закрывать окно. Вот как это делается:

```
<A HREF="" onClick="self.close()">Щелкните, чтобы закрыть</A>
```

Это обычная ссылка `HREF`, которая никуда не ведет. Видите пустые кавычки? Задание ссылки таким образом позволяет избежать загрузки страницы. Закрывает окно команда `onClick="self.close()"`.

`self` (само, себя) — это свойство может относиться к любому объекту. В нашем случае это свойство окна. Команда `close` (закрыть) закрывает окно.

Еще один вопрос

Допустим, что требуется открыть окно по команде, а не когда пользователь заходит на страницу. Вот как это можно сделать:

```
<A HREF="les11.htm" onClick="window.open('opened.html', 'joe',
config='height=300,width=300')">Щелкните, чтобы открыть 'joe'</A>
```

Это ссылка `HREF`, которая направлена на саму себя. Команда `onClick` делает работу, а параметры содержатся в скобках `()`.

В следующем уроке мы поговорим о том, как из одной страницы сделать две.

Задание

Здесь не были показаны в действии все доступные функции, так что в качестве задания напишите сценарий, который откроет новое окно со всеми характеристиками. Пусть оно будет размером `300` на `500` пикселей и содержит две ссылки:

одна откроет новую страницу в главном окне; вторая откроет новую страницу в том же окне.

Страница, которая откроется в том же маленьком окне, должна содержать ссылку, закрывающую окно.

И сделайте фон страницы желтым (`ffff00`).

Возможное решение

```
<SCRIPT type="text/javascript">

window.open ('opened2.html', 'newwin',
config='height=300,width=500,toolbar=no,
menubar=yes,scrollbars=no,resizable=no,
location=no,directories=yes,status=no')

</SCRIPT>
```

Использованы все характеристики. Ссылки внутри нового окна написаны так же, как и в материалах урока.

Открытие окна с помощью функции

Основные функции для манипуляции с окнами.

Концепция

В Уроке 11 новое окно открывалось с помощью команды `window.open`. В окне выводился другой документ HTML, который был указан в скобках.

Сейчас мы попробуем создать функцию, которая откроет новое окно, — причем и новое окно, и все его содержимое будет содержаться в том же документе HTML. То есть, в буквальном смысле слова мы вложим две страницы в одну.

Сценарий

```
<SCRIPT type="text/javascript">
function openindex()
   {
var OpenWindow=window.open("", "newwin", "height=300,width=300");
OpenWindow.document.write("<HTML>");
OpenWindow.document.write("<TITLE>Новое окно</TITLE>");
OpenWindow.document.write("<BODY BGCOLOR='00ffff'>");
OpenWindow.document.write("<CENTER>");
OpenWindow.document.write("<font size=+1>Новое окно</font><P>");
OpenWindow.document.write("<a href='http://www.intuit.ru' target='main'>
Эта ссылка<BR> откроется в основном окне</a><p>");
OpenWindow.document.write("<P><HR WIDTH='60%'><P>");
OpenWindow.document.write("<a href='' onClick='self.close()'>
Эта ссылка закроет окно</a><p>");
OpenWindow.document.write("</CENTER>");
OpenWindow.document.write("</HTML>");
}
</SCRIPT>
```

...и в команде `BODY`:

```
onLoad="openindex()"
```

Результат работы сценария

Результат такой же, что и в прошлом уроке: открылось окно того же размера с теми же двумя ссылками. Разница в том, что все это было написано на одной странице.

Разбор сценария

Главная часть сценария, содержащая функцию, помещается между тегами `<HEAD>` и `</HEAD>`, как большинство функций.

По обычной схеме для функции задается имя `openindex()`. Затем следуют фигурные скобки. Теперь подходим к основному моменту. Создаем переменную `OpenWindow`, под которой скрывается команда `window.open()`. Она выглядит следующим образом:

```
var OpenWindow=window.open("", "newwin", "height=300,width=300");
```

Формат знакомый. Единственная разница в том, что не указан URL. Видите пустые парные кавычки? Они говорят браузеру, что он должен искать в сценарии информацию о новом окне, — точно так же, как и в случае отсутствия URL в команде, которая закрывает окно. Оно бы не закрылось, если бы начала загружаться новая страница. То же самое и тут. Браузер стал бы загружать новую страницу, а не выполнять сценарий.

Теперь начинаем создавать страницу HTML, которая будет в новом окне. Вот первая строка текста:

```
OpenWindow.document.write("<HTML>")
```

Команда говорит, что строка текста должна быть записана в документ переменной `OpenWindow` (новое окно) .

Посмотрите на сценарий. Каждая новая строка следует той же схеме. Можно написать сотню строк, создающих законченную страницу. Наш сценарий совсем небольшой, так как это учебный пример.

Помните: когда вы пишете HTML внутри команды `document.write`, вместо двойных кавычек с подкомандами ставьте одинарные. Иначе будет ошибка.

Наконец обработчик событий `onLoad` в команде `BODY` вызывает функцию.

Задание

Написать функцию, которая открывает окно. Документ, который появится в окне должен иметь зеленый фон и заголовок `TITLE`: "Привет, "имя пользователя", вот твое окно!" Имя пользователя можно узнать с помощью запроса (`prompt`). Разумеется, добавьте еще ссылку, которая закроет окно.

Возможное решение

Сценарий можно взять прямо из сегодняшнего урока и внести несколько изменений:

в начале функции добавить команду `prompt` ; разбить команду `TITLE` на три части и внести в нее переменную `name` ; цвет фона поменять на зеленый.

Вот готовый сценарий:

```
<SCRIPT type="text/javascript">
function openindex()
{

var name=prompt("Как вас зовут?","Напишите здесь")

var OpenWindow=window.open("", "newwin", "height=300,width=300,status=ye

OpenWindow.document.write("<HTML>")
OpenWindow.document.write("<TITLE>")
OpenWindow.document.write("Привет, " +name+ "! Вот ваше окно!")
OpenWindow.document.write("</TITLE>")
```

```
OpenWindow.document.write("<BODY BGCOLOR='green'>")
OpenWindow.document.write("<CENTER>")
OpenWindow.document.write("<h2>Новое окно</h2>")
OpenWindow.document.write("<a href='' onClick='self.close()'>
  Эта ссылка закроет окно</a>")
OpenWindow.document.write("</CENTER>")
OpenWindow.document.write("</BODY>")
OpenWindow.document.write("</HTML>")
}
</SCRIPT>

<body bgcolor="xxxxxx" onLoad="openindex()">
```

Метод 'Confirm' (Введение в if и else)

Основы логического разветвления сценариев.

Концепция

Команда `confirm` (подтвердить) действует очень похоже на метод `alert`, но добавляет в диалоговое окно кнопку "Отмена" (Cancel). И то, и другое — методы.

Одна команда сама по себе многого не дает. Нет никакой разницы, что вы выбираете — "ОК" или "ОТМЕНА". Но стоит добавить функции `if` (если) и `else` (иначе), и можно создать интересные эффекты.

Сценарий

Прежде всего посмотрим на базовый формат:

```
<SCRIPT type="text/javascript">

confirm("Уверены, что хотите войти?")

</SCRIPT>
```

Выглядит знакомо. То же самое, что и `alert`, кроме слова `confirm`. Как видите, сценарий делает не очень много. Но вот та же команда с некоторыми добавлениями:

```
<SCRIPT type="text/javascript">

if (confirm("Уверены, что хотите посетить INTUIT?") )

{
parent.location='http://www.intuit.ru/';
alert("Счастливого пути");
}

else
```

```
{
alert("Тогда оставайтесь");
}

</SCRIPT>
```

Результат работы сценария

Это уже кое-что. Выводится ссылка с вопросом. Только на этот раз, если нажать "OK", то произойдет переход по ссылке, а если щелкнуть на "Отмена", то останетесь на странице.

Давайте разберемся, как это делается.

Разбор сценария

У вас есть выбор

Во-первых, сценарий говорит:

```
if (confirm("Уверены, что хотите посетить INTUIT?") )
```

Это значит Если (Здесь можно сделать выбор).

В нашем случае confirm предлагает варианты: "OK" или "Отмена". Можно считать их ответами Да и Нет. Обратите внимание на скобки. После команды `if` всегда идут круглые скобки, но, как известно, команда confirm тоже требует скобок. Следовательно, берем две пары скобок, одна внутри другой.

Сразу же после этого идут команды, выполняемые при каждом варианте ответа. Обратите внимание на фигурные скобки `{}`. Зачем? Потому что в действительности это функции. Первая из них показывает, что должно произойти, если пользователь выберет OK (или Да).

```
{
parent.location='http://www.intuit.ru/';
alert("Счастливого пути");
```

```
}
```

Если помните, несколько уроков назад говорилось, что `parent.location` является командой, создающей ссылку. Дальше идет обыкновенная команда `alert`. Не забудьте про точку с запятой в конце строк.

А если выбрать отмену?

Мы уже знаем, что если выбрать OK, то выполнится функция, следующая непосредственно за оператором `if` (если). "Cancel" (Отмена) — другой выбор. Видите, сразу после фигурной скобки идет команда `else` (иначе), как бы "если нет". И тогда следующий текст...

```
else
{
alert('Тогда оставайтесь');
}
```

...означает: если нет, тогда послать сообщение и не менять страницу.

Все это вместе и дает пользователю возможность выбора: входить или не входить.

Это самые основы использования `if` и `else`. Позже команде `if` мы посвятим целый урок. Она того заслуживает, как вам кажется?

Задание

Не пугайтесь, ничего сложного. Преобразуйте рассмотренный в этом уроке сценарий в функцию. И сделайте так, чтобы при отмене (Cancel), кроме окна, еще появлялась какая-нибудь надпись в строке состояния.

Можно также попробовать сделать так, чтобы при выборе OK страница перехода открывалась в новом окне.

Возможное решение

В разделе заголовка HEAD помещаем следующий код.

```
<SCRIPT LANGUAGE="javascript">

function go()
{
if (confirm("Хотите на INTUIT?") )
{
parent.location='http://www.intuit.ru';
alert("Счастливого пути");
}
else
{
alert("Ладно, оставайтесь");
defaultStatus='Что сделано, то сделано';
}}
</SCRIPT>
```

...и в команду <BODY>:

```
<BODY onLoad="go()">
```

Процесс на самом деле достаточно простой:

- создается имя функции, сценарий копируется и помещается в фигурные скобки;
- в раздел `else` добавляется команда `defaultStatus='Что сделано, то сделано'`, которая выводит текст в строку состояния;
- функция запускается командой `onLoad` в строке `BODY` ;
- если хотите, чтобы ссылка открылась в новом окне, то нужно изменить только два слова. Замените `parent.location` на `window.open`, и все готово.

Математические вычисления

Математические функции и их применение.

Концепция

В этом уроке мы узнаем, как производить математические вычисления с помощью JavaScript. Если вам уже приходилось заниматься программированием, то все будет знакомо. Если нет, не паникуйте. Все очень просто!

Сценарий

```
<BODY>
  <SCRIPT type="text/javascript">
    var numsums = 10 + 2
        alert("10 + 2 равно " + numsums)
    var x = 10
        alert("десять — это " + x)
    var y = x * 2
        alert("10 X 2 = " + y)
    var z = "Привет " + "Пока"
   alert(z)
  </SCRIPT>
</BODY>
```

Результат работы сценария

Попробуйте определить значение каждой из переменных перед выполнением сценария.

При выполнении сценарий последовательно выводит ряд окон, содержащих:

10 + 2 равно 12

десять — это 10

10 X 2 = 20

Привет Пока

Разбор сценария

```
<BODY>
  <SCRIPT type="text/javascript">
    var numsums = 10 + 2
        alert("10 + 2 равно " + numsums)
    var x = 10
        alert("десять — это " + x)
    var y = x * 2
        alert("10 X 2 = " + y)
    var z = "Привет " + "Пока"
    alert(z)
  </SCRIPT>
</BODY>
```

В сценарии задается переменная `numsums`. Видите, она равна 12 (10+2)? Затем эта переменная используется в методе `alert` и выводит "10 + 2 = переменная" или 12.

Другая переменная, `x`, задается равной `10`. Затем метод `alert` выводит это значение.

Следующая переменная, `y`, равна переменной `x`, умноженной на `2`. Дважды десять — двадцать, не правда ли? Этот результат затем выводится в окне `alert`.

Наконец создается переменная `z`, которая показывает, что можно соединять текст с помощью знака сложения. Затем эта переменная выводится с помощью метода `alert`.

Посмотрите еще раз, как работает скрипт.

Основные моменты:

- Переменные начинаются со слова `VAR` (от variable, переменная), затем идет имя, знак `=` и значение переменной. `VAR` можно не писать, но лучше не отказываться от него, пока не появится некоторый опыт.
- Имя переменной может состоять из одного или нескольких символов (буквы, цифры, символ подчеркивания). Но лучше использовать содержательные имена.
- Имена переменных различают регистр! То есть, `Xvar` и `xvar` — это два разных имени переменных.
- Допустимая длина имени переменной существенно различна для разных браузеров. В целях безопасности берите имена не больше 10 символов. Не используйте в именах пробелы.
- Значение, присваиваемое текстовой переменной, ставится в кавычки. Числовые переменные не ставятся в кавычки, иначе сценарий воспримет их как текст с числовым значением `0`!
- Операции сложения, вычитания, умножения и деления обозначаются знаками `+`, `-`, `*`, и `/` соответственно.
- Знак плюс (`+`) выполняет две задачи: сложение чисел или соединение вместе двух строк текста (например, `"Joe" + " Burns"` будет `"Joe Burns"`).
- Все языки программирования имеют зарезервированные слова, например, названия команд. В любой книге можно найти их перечень. Использование зарезервированных слов в качестве имен переменных будет приводить к ошибкам.
- Если необходимо, применяйте в именах переменных вместо пробела знак подчеркивания `_ user_name`.

Задание

Перепишите приведенный выше код JavaScript в виде функции. Можно, при желании, изменить некоторые математические операции, например, на деление. Включите в тело документа HTML приветственное сообщение. Используйте для выполнения функции событие `onLoad`.

Возможное решение

```
<html>
<head>
  <SCRIPT type="text/javascript">
   function vars()
   {
     numsums=10 + 2
     alert("10 + 2 is " + numsums)
     var x = 10
     alert("ten is " + x)
     y = x * 2
    alert("10 X 2 = " + y)
     z = "Привет " + "Пока"
    alert(z)
   }
  </SCRIPT>
</head>
<BODY OnLoad="vars()">
<h1>Добро пожаловать на мою страницу</h1>
</body>
</html>
```

Мы будем заниматься переменными очень много по мере изучения новых команд Java Script. Главное, пытайтесь понять, что делаете, а не копируйте автоматически.

Изменение изображения с помощью события onMouseOver

Работа с изображениями. Динамическое изменение изображений.

Концепция

В данном примере будут рассмотрены дополнительные возможности использования событий `onMouseOver` и `onMouseOut`. Как мы говорили ранее, любое событие может запускать на выполнение функцию или оператор JavaScript. Вспомните команду `onLoad` в теле документа HTML, которая вызывает код JavaScript из заголовка `HEAD`.

Представленные здесь два события происходят, когда указатель мыши перемещается на ссылку или смещается со ссылки. В Уроке 4 эти обработчики событий использовались для создания эффекта появления текста в строке состояния.

Еще раз обратите внимание на то, что теги `<SCRIPT>` и `</SCRIPT>` не требуются. Обработчики событий `onMouseOver` и `onMouseOut` встраиваются в тег HTML `<A HREF>`. Также отметим, что для отключения вывода рамки вокруг изображения в теге `<IMG SRC>` включен атрибут `BORDER="0"`.

Сценарий

```
<A HREF="http://www.intuit.ru"
onMouseOver="document.pic1.src='on.gif'"
onMouseOut="document.pic1.src='off.gif'">
<IMG SRC="off.gif" BORDER=0 NAME="pic1">
</a>
```

Результат работы сценария

На странице выводится изображение off.gif. Если навести на изображение указатель мыши, то изображение изменится на on.gif. При смещении указателя мыши с изображения возвращается изображение

off.gif.

Разбор сценария

```
<A HREF="http://www.intuit.ru"
onMouseOver="document.pic1.src='on.gif'"
onMouseOut="document.pic1.src='off.gif'">
<IMG SRC="off.gif" BORDER=0 NAME="pic1">
</a>
```

Так как обработчики события были достаточно хорошо рассмотрены ранее, то этот сценарий не представляет трудностей. Когда курсор уходит с изображения, появляется off.gif. Когда указывает на изображение, появляется on.gif.

Обратите внимание, что команда `IMG` связана с обработчиками `onMouse` в команде `HREF` через команду `NAME="pic1"`. Это необходимо для связи команд.

Основные моменты:

1. `onMouseOver` и `onMouseOut` различают регистр. Нельзя менять заглавные и строчные буквы.
2. Так как необходимо ставить кавычки после `onMouseOver=` и `onMouseOut=`, то название файла *.gif берется в одинарные кавычки, а не в двойные.
3. `document.pic1.src` следует иерархии объектов, о которой говорилось в Уроке 8. `document` относится к текущему объекту документа HTML, а `pic1` — это имя объекта изображение (имя можно придумать какое угодно). `src` (источник) — это свойство объекта изображение, которое указывает файл изображения.
4. В этом примере `onMouseOver` меняет источник изображения на on.gif.
5. `OnMouseOut` заставляет объект изображение вывести off.gif.
6. Имейте в виду, что для наилучшего эффекта картинки должны быть одинакового размера.

Задание

Создайте страницу HTML. Разместите все по центру страницы. Используйте тег `H1` со своим именем. Под ним поместите изображение Bubble1.gif. Когда курсор мыши укажет на это изображение, оно должно измениться на изображение Bubble2.gif. Когда курсор мыши сместится с этой ссылки, снова должно появиться изображение Bubble1.gif.

Возможное решение

Например, следующий код:

```
<html>
  <head>
  </head>
  <body bgcolor="white">
  <center>
  <h1>Иван Иванович</h1>
    <p>
  <a href="" onMouseOver="document.pic1.src='bubble2.gif' "
    onMouseOut="document.pic1.src='bubble1.gif'">
  <img src="bubble1.gif" border=0 name="pic1"></a></p>
  </center>
  </body>
  </html>
```

Существует множество разных трюков с `onMouseOver` и `onMouseOut`. Прежде чем двинуться дальше, в следующем уроке рассмотрим еще один пример.

Изменение изображения с помощью функции

Динамическое изменение изображений с помощью функций.

Концепция

Вот еще один пример использования `onMouseOver` и `onMouseOut`. На этот раз вместо включения оператора JavaScript для смены картинки в тег `<A HREF>` обработчики событий `onMouseOver` и `onMouseOut` вызывают функцию.

В общем, когда нужна только одна команда JavaScript, можно включить ее в тег HTML `<A HREF>`. Для нескольких операторов JavaScript больше подходит функция. В реальном мире на странице часто требуется многократное изменение изображения с помощью JavaScript.

Сценарий

```
<HTML>
<HEAD>
  <title> Пример JavaScript </title>
<SCRIPT type="text/javascript">
function up()
{
   document.mypic.src="up.gif"
}
function down()
{
   document.mypic.src="down.gif"
}
</SCRIPT>
</HEAD>
  <BODY>
    <CENTER>
    <h2>Пример анимации</h2>

<A HREF="http://www.intuit.ru"
  onMouseOver="up()"  onMouseOut="down()">
```

```
<IMG SRC="down.gif" NAME="mypic" BORDER=0></A>
  </BODY>
</HTML>
```

Результат работы сценария

Пример анимации с помощью `OnMouseOver` и `OnMouseOut`.

Если быстро двигать курсором мыши на изображении, то возникает ощущение ожившего изображения — анимации.

Разбор сценария

Не забывайте про регистр символов при записи `OnMouseOver` и `OnMouseOut`.

Обратите внимание, что в сценарии созданы две функции.

```
<SCRIPT type="text/javascript">
function up()
{
document.mypic.src="up.gif"
}
function down()
{
document.mypic.src="down.gif"
}
```

Функции выполняют то же, что и команды в прошлом уроке. Помните иерархию объектов? Сначала документ, потом имя, присвоенное объекту, и наконец `SRC`. Функции названы `up()` и `down()`.

Теперь посмотрим на вызов функции:

```
<A HREF="http://www.intuit.ru"
onMouseOver="up()" onMouseOut="down()">
<IMG SRC="down.gif" NAME="mypic"
BORDER=0></A>
```

Формат почти такой же, как и использованный в Уроке 15, но здесь вместо включения оператора в команду `HREF` вызывается функция.

Этот пример по-прежнему имеет один оператор JavaScript. Вспомните, что обычно функции используются для объединения нескольких операторов. Мы хотели просто сделать пример коротким.

Если вы решите использовать многократную смену картинок с помощью JavaScript, помните, что для каждого случая нужно создавать имя новой функции и соединять определенные изображения с этими функциями, изменяя также атрибут `NAME`. Например: вы хотите поместить на страницу еще одну такую же меняющуюся картинку. Для этого создаем две новые функции, копируя предыдущие две функции и добавляя к именам цифру 2 (обратите на это внимание в коде ниже). Затем необходимо изменить в каждом случае атрибут `NAME` (снова обратите внимание на код ниже), поэтому изменяем имя на `mypic2`. Не забудьте изменять его везде, где оно появляется. Получаем примерно следующее в разделе заголовка `HEAD`:

```
<SCRIPT LANGUAGE="JavaScript">
function up()
{
  document.mypic.src="up.gif"
}
function down()
{
  document.mypic.src="down.gif"
}

function up2()
{
  document.mypic2.src="up.gif"
}

function down2()
{
  document.mypic2.src="down.gif"
}
```

```
</SCRIPT>
```

... и примерно следующее для вызова двух различных изображений:

```
<A HREF="http://www.intuit.ru"
      onMouseOver="up()"
      onMouseOut="down()">
   <IMG SRC="down.gif" NAME="mypic"
BORDER=0></A>

<a href="http://www.intuit.ru"
      onMouseOver="up2()"
      onMouseOut="down2()">
   <IMG SRC="down.gif" NAME="mypic2"
BORDER=0></A>
```

Видите, как новые функции связаны с новыми именами? Делайте это каждый раз при добавлении новой меняющейся картинки. Если требуется три случая, то добавьте цифру 3 везде, где выше добавлена 2. Если четыре, добавьте 4, и т.д.

Задание

Переделайте последнее задание, использующее Bubble1.gif и Bubble2.gif. Создайте две функции для смены этих изображений.

Возможное решение

```
<html>
 <head>
 <SCRIPT type="text/javascript">
   function swap1for2()
     {  document.pic1.src="bubble2.gif" }
   function swap2for1()
     { document.pic1.src="bubble1.gif" }
 </SCRIPT>
 </head>
 <body bgcolor="white">
```

```
<center>
<h1> Иван Иванович</h1>
<a href="" onMouseOver="swap1for2() "
  onMouseOut="swap2for1()">
<img src="bubble1.gif" border=0 name="pic1"> </a>
</center>
</body>
</html>
```

Вызов функции в формы

Создание функций для обработки данных пользователя в формах.

Концепция

Цель этого урока — познакомить вас с применением JavaScript в формах. Рассматриваемый сценарий использует форму для выбора цвета фона, голубого или розового. Отметим, что выбор цвета делается с помощью кнопок формы.

Формы всегда начинаются тегом `<FORM>` и заканчиваются тегом `</FORM>`. В этом ничего нового, простой HTML.

Сценарий

Пример ниже снова показывает весь документ HTML:

```
<html>
<head>
<SCRIPT type="text/javascript">
function newcolor(color)
{
  alert("Вы выбрали " + color)
  document.bgColor=color
}
</SCRIPT>
</HEAD>
<BODY>
<p>Выберите цвет фона</p>
<FORM>
 <INPUT TYPE="button" VALUE="Голубой"
      onClick="newcolor('lightblue')">
 <INPUT TYPE="button" VALUE="Розовый"
      onClick="newcolor('pink')">
</FORM>
</BODY>
</HTML>
```

Результат работы сценария

На странице выводятся две кнопки с надписями "Голубой" и "Розовый". При нажатии на любую кнопку цвет фона меняется в соответствии с указанным на кнопке цветом.

Разбор сценария

Пришло время для нового термина! Литерал является значением (VALUE), которое не изменяется. Литерал может быть числом, именем или любой случайной последовательностью чисел и имен. Помните только о том, что литерал невозможно изменить.

Еще один новый термин! Строка является любой последовательностью букв или цифр в одиночных или двойных кавычках. Таким образом следующий фрагмент сценария:

```
onClick="newcolor('lightblue')"
```

... определяет строку литерал "lightblue". Все понятно? Отлично.

Вот снова сценарий и элементы ввода:

```
function newcolor(color)
{
 alert("Вы выбрали " + color)
 document.bgColor=color
}

<FORM>
 <INPUT TYPE="button" VALUE="Голубой"
     onClick="newcolor('lightblue')">
 <INPUT TYPE="button" VALUE="Розовый"
     onClick="newcolor('pink')">
</FORM>
```

Обратите внимание, мы передаем в функцию newcolor() (новый цвет) строку литерал, 'lightblue' или 'pink'. Она находится в одинарных кавычках, потому что имя функции стоит в двойных.

Когда вы нажимаете кнопку, строка в скобках (`'pink'` или `'lightblue'`) передается в функцию `newcolor()`.

Функция ждет, пока поступит необходимая ей информация. Вспомните, что во всех функциях до сих пор скобки были пустые. Функции имели все необходимые данные. В данном случае дополнительная информация поступает в функцию, когда пользователь нажимает на кнопку. Кнопка содержит ту же функцию, только теперь у нее есть необходимые данные, то есть цвет.

Форма передает цвет двум элементам в разделе `<SCRIPT>`: методу `alert` и свойству `document.bgColor`. Получив все данные, функция вступает в действие: всплывает окно и меняется цвет фона.

Не запутайтесь: атрибут `VALUE` (значение) в команде `INPUT` не является свойством JavaScript, он выводит текст на кнопку. Он не влияет на свойства JavaScript.

Задание

Посмотрим, хорошо ли вы помните материал предыдущих уроков. Перепишите скрипт так, чтобы, открываясь, страница предлагала пользователю ввести имя. При выборе цвета должно всплывать окно со словами "Эй, (имя)! Вы выбрали (цвет)...".

Возможное решение

Чтобы добиться нужного результата, вставьте запрос prompt перед функцией, а потом результат запроса — в команду `alert`. Весь документ выглядит следующим образом :

```
<html>
<head>
<SCRIPT type="text/javascript">
var user_name = prompt ("Можно узнать, как Вас зовут?","Ваше имя");
function newcolor(color)
{
alert("Эй, " + user_name + "! Вы выбрали " + color)
```

```
document.bgColor=color
}
</SCRIPT>
</head>
<body bgcolor="white">
<center><h1>Задайте фон страницы</h1>
</center>
<form>
  <input type="button" value="Оранжевый" onClick="newcolor('orange')">
  <input type="button" value="Салатовый" onClick="newcolor('lightgreen')">
</form>
</body>
</html>
```

Поля формы и свойство value

Динамическое изменение содержимого форм.

Концепция

Сегодня мы продолжим начатое в Уроке 17. Будем передавать в функцию данные, которые пользователь введет в поле формы. Затем эти данные будут использованы для поиска в Yahoo.

Сценарий

```
<SCRIPT type="text/javascript">

function Gofindit(){
 var searchfor = document.formsearch.findthis.value;
{
 var FullSearchUrl =
"http://www.yahoo.com/bin/query?p=" + searchfor ;
location.href = FullSearchUrl;
}}

</SCRIPT>

<FORM NAME="formsearch" action="">
Найдите в Yahoo:
<INPUT NAME="findthis" SIZE="40" TYPE="text">
<INPUT TYPE="button" VALUE="Искать"
onClick="Gofindit()">
</FORM>
```

Результат работы сценария

На странице выводится строка "Найдите в Yahoo:", поле ввода и кнопка с надписью "Искать".

Разбор сценария

Этот сценарий снова требует четкого понимания иерархии объектов.

1. Во-первых, создаем функцию с переменной `searchfor` (искать) под названием `formsearch`, внутри элемента `findthis` (найти), который обладает свойством `value` (значение). Она будет результатом чего-то происходящего в объекте `document`.
2. Вторую функцию помещаем внутри первой. Видите вторую пару {фигурных скобок}?
3. Для второй функции создаем еще одну переменную `FullSearchUrl`, которая представляет собой адрес поисковой машины Yahoo плюс значение переменной `searchfor`, полученное через команду `document.formsearch.find.value`.
4. Наконец, `location.href` приравнивается переменной `FullSearchUrl`. После выполнения функции пользователь попадет на итоговую страницу поиска.
5. Теперь переходим к командам формы. Их две: текстовое поле (`TEXT`), куда пользователь вводит свой запрос, и кнопка, запускающая функцию.
6. Обратите внимание, что форма в целом называется `formsearch`. Помните, что мы говорили об иерархии объектов?
7. Затем для текстового поля задаем имя `findthis`. Опять иерархия. Видите, как мы идем от большого к малому?
8. Дальше соединяем кнопку с командой `onClick`, которая запускает функцию.
9. Наконец заканчиваем форму командой `</FORM>`. Готово.

Задание

Измените сценарий так, чтобы он вызывал другую поисковую систему. И еще, пусть при отправке запроса появляется окошко с надписью "Сейчас поищем..."

Возможное решение

Поиск в Яndex

Если известен адрес поисковой машины, тогда никаких особых трудностей не будет. Но как его узнать? Идите на сайт, начните поиск и скопируйте адрес из адресной строки.

Раз нам нужно, чтобы окно всплывало до завершения поиска, убедитесь, что оно стоит у вас первым. Вот сценарий, который выполняет эту работу:

```
<SCRIPT type="text/javascript">

function Gofindit()
{
alert("Сейчас поищем...");
var search = document.formsearch.find.value;
{
var searchUrl = "http://www.yandex.ru/yandsearch?ctgl=11657&text=" + search;
location.href = searchUrl;
}}

</script>


<form name="formsearch">
Поискать в Яndex:
<input type="button" value="Найти" onClick="Gofindit()">
</form>
```

Передача данных в функцию

Базовые аспекты обработки данных функциями.

Концепция

Продолжим разговор о взаимодействии форм и JavaScript. Здесь становится особенно важно понимание иерархии объектов и работы функций.

Обычно JavaScript в соединении с формами используется для проверки ввода данных. Мы еще поговорим на эту тему.

Сценарий

Здесь снова представлен весь документ HTML, чтобы показать размещение отдельных элементов.

```
<HTML>
<HEAD>
<SCRIPT type="text/javascript">
function doit()
{
  alert("document.myform.fname.value — это "
+ document.myform.fname.value)
  var greeting="Привет "
  alert(greeting + document.myform.fname.value
+ " " + document.myform.lname.value)
  alert("Длина имени "
+ document.myform.fname.value.length)
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="myform" action="">
  Ваше имя?
   <INPUT TYPE="text" NAME="fname"><p>
  Ваша фамилия?
```

```
    <INPUT TYPE="text" NAME="lname"><p>
    <INPUT TYPE="button"
      VALUE="Отправить"
        onClick="doit()">
  </FORM>
  </BODY>
  </HTML>
```

Результат работы сценария

На странице выводится два поля ввода с просьбой ввести имя и фамилию и кнопка "Отправить". После нажатия на кнопку введенные данные отображаются в последовательно появляющихся окнах `alert`.

Разбор сценария

Начнем с элементов формы:

```
<FORM NAME="myform" action="">
  Ваше имя:
     <INPUT TYPE="text" NAME="fname"><p>
  Ваша фамилия:
     <INPUT TYPE="text" NAME="lname"><p>
  <INPUT TYPE="button" VALUE="Отправить" onClick="doit()">
  </FORM>
```

Видите, мы дали форме имя `myform`. Текстовое поле ввода для имени пользователя названо `fname`, а поле для фамилии — `lname`. Теперь у каждого элемента есть имя.

Данные, которые введет пользователь, станут значением (`value`) соответствующих текстовых полей. Понятно? Тому, что написано в поле `fname`, будет присвоено имя `fname`.

Когда пользователь нажмет на кнопку (обработчик события `onClick`), выполнится функция `doit()`.

Теперь посмотрим на функцию:

```
function doit()
{
 alert("document.myform.fname.value — это  "
+ document.myform.fname.value)
 var greeting="Привет, "
 alert(greeting + document.myform.fname.value + " "
+ document.myform.lname.value)
 alert("Длина имени "
+ document.myform.fname.value.length)
}
```

Никакие параметры, как на предыдущих уроках, не передаются. Видите, в скобках функции `doit()` ничего нет. Но по иерархическим командам понятно, что функция использует данные, введенные в форму.

Мы следуем иерархии объектов: за объектом документ следует объект форма (на него указывает имя формы, `myform`), за ним — объект поле формы (на него указывает имя поля, `fname`), за ним — свойство значение (`value`). Без свойства `value` данные, переданные пользователем, не попали бы в иерархическую команду.

Дальше идет переменная *`greeting`* (приветствие). Содержимое *`greeting`* выводится с помощью команды `alert(`*`greeting`*`)`.

Когда пользователь нажимает на кнопку, всплывает окно с его именем.

Второе окно включает в себя переменную *`greeting`*. Появляется надпись: "Привет, (имя) (фамилия)", которая составлена с помощью данных, полученных через форму. Еще раз обратите внимание на `value`.

Наконец всплывает третье окно с неким текстом и вызывает команду: `document.myform.fname.value.length`. Эта команда выводит длину (`length`) слова, введенного в поле формы `fname`. Если `fname` содержит имя "Коля", то длина равна 4. Команда `length` следует за `value`. Таким образом она подсчитает количество букв в тексте. `length` — это тоже свойство.

Задание

Составьте документ HTML с формой `aform`. В ней должно быть два текстовых поля (одно для города, другое для страны) и кнопка. Напишите функцию с переменной, которая содержит слова "Мне нравится ". Когда пользователь нажмет на кнопку, должно всплывать окно со следующей надписью:

Мне нравится город (страна).
(по результатам тех данных, которые пользователь вводит в форму)

Покажите длину (`length`) названия города.

Возможное решение

Существуют разные решения, например, такое:

```
<html>
 <head>
 <SCRIPT type="text/javascript">
 function doit()
 {
  var greeting="Мне нравится "
  alert(greeting + document.aform.city.value
      + " (" + document.aform.state.value +").")
  alert
   ("В названии вашего города " + document.aform.city.value.length
      + " букв.")
 }
 </script>
 </head>
 <body >

 <form name="aform">
  В каком городе вы живете?
     <INPUT TYPE="text" NAME="city">
  В какой стране?
     <INPUT TYPE="text" NAME="state">
  <input type="button"
       value="Отослать" onClick="doit()">
 </form>
```

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

Легко видеть, что имена полей ввода из примера были изменены на `city` и `state`. Потом была убрана первая команда `alert`, а все остальное осталось почти без изменений.

Создание случайных чисел

Методы создания случайных событий и работа с ними.

Концепция

В этом примере рассматриваются случайные числа. Для генерации случайных чисел JavaScript использует дату и время.

Обратите в сценарии внимание на следующее: число после знака % является ограничивающим числом. Пример ниже выбирает случайное число между 1 и 10.

Сценарий

```
<HTML>
<HEAD>
<SCRIPT type="text/javascript">
function rand()
      {
         var now=new Date()
         var num=(now.getSeconds())%10
         var num=num+1
         alert(num)
      }
</SCRIPT>
</HEAD>
<BODY>
 <h1>Случайные числа</h1>
 <form>
  <INPUT TYPE="button"
  VALUE="Случайные числа от 1 до 10"
  onClick="rand()">
 </FORM>
</BODY>
</HTML>
```

Результат работы сценария

На странице выводится кнопка с надписью "`Случайные числа от 1 до 10`", при нажатии на которую появляется окно со случайным числом от 1 до 10.

Разбор сценария

Начнем в этот раз с функции:

```
function rand()
{
var now=new Date()
var num=(now.getSeconds())%10
var num=num+1
alert(num)
}
```

Выбор случайного числа осуществляется в несколько шагов.

1. Во-первых, создаем функцию. Наша называется `rand()`.
2. Потом создаем переменную для метода `new Date()`.
3. Создаем еще одну переменную, `num`. Она содержит метод `getSeconds()`, так как в данном случае мы используем секунды для выбора случайного числа.

JavaScript, как и многие другие компьютерные языки, начинает отсчет с нуля. Поэтому элемент `%10` говорит JavaScript, что случайное число будет выбираться из чисел от 0 до 9.

Оператор `%` возвращает остаток от деления. Предположим, что функция `getSeconds()` вернула значение 20 секунд. При делении на 10 получаем остаток 0. Сценарий возвращает 0. Пусть число секунд равно 12. Остаток при делении на 10 будет равен 2.

Прибавляя к случайному числу единицу (`num=num+1`), мы получаем числа не от 0 до 9, а от 1 до 10.

Наконец `alert` выводит число.

Теперь элемент, запускающий функцию:

```
<form action="">
  <INPUT TYPE="button"
  VALUE="Случайное число от 1 до 10"
  onClick="rand()">
</form>
```

Это обычная кнопка, которая запускает выполнение приведенной выше функции.

Задание

Напишите программу JavaScript, в которой пользователь нажимал бы кнопку в форме, а программа выводила бы случайное число от 0 до 4 со словами: "Ваше случайное число: "x".

Возможное решение

```
<html>
    <head>
    <SCRIPT type="text/javascript">
    function rand()
       {
         now=new Date()
         num=(now.getSeconds())%5
         alert("Ваше случайное число: " + num)
       }
    </script>
    </head>
    <body>
       <h1>Случайные числа</h1>
       <form>
     <input type="button" value="Вывод случайного числа между 0 и 4"
    onClick="rand()">
      </form>
```

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

Необходимо изменить число после знака процента на `5` и добавить немного текста в команду `alert`, и все будет готово.

Оператор if и ветвление

Логическое разветвление программ. Условный оператор.

Концепция

Этот пример знакомит с оператором `IF` (если), который позволяет решить, что делать, "если" выполняется какое-то условие. Программа спрашивает пользователя, любит ли он спорт. Если он отвечает `"да"`, то программа отвечает `"Я тоже люблю спорт"`. Если пользователь говорит `"нет"`, то программа отвечает `"Я тоже ненавижу спорт"`. Это немного глуповато, но для краткого знакомства вполне подходит.

Отметим, что если пользователь вводит что-то отличное от `"да"` или `"нет"`, то программа выводит сообщение `"Отвечайте да или нет"`.

За оператором `IF` следует условие и указание, что делать, если оно верно. Можно проверять одно условие или несколько. Программа знает, где начинаются и кончаются исполняемые после условия операторы, потому что они заключены в {фигурные скобки}.

Сценарий

```
<HTML>
<HEAD>
 <SCRIPT type="text/javascript">
  function askuser()
  {
   var answer="   "
   var statement="Отвечайте да или нет"
   var answer=prompt("Вы любите спорт?")
   if ( answer == "да")
    {statement="Я тоже люблю спорт!"}
   if(answer == "нет")
    {statement="Я тоже ненавижу спорт!"}
```

```
    alert(statement)
     }
   </SCRIPT>
 </HEAD>
 <BODY>
  <h1>Деятельность</h1>
   <FORM action="">
    <INPUT TYPE="button" VALUE="Нажми здесь!"
      onClick="askuser()">
   </FORM>
 </BODY>
 </HTML>
```

Результат работы сценария

На странице выводится кнопка со словами "`Нажми здесь!`", при нажатии на которой появляется окно с вопросом и полем ввода. В зависимости от ответа выводятся различные сообщения.

Разбор сценария

Начнем с кнопки:

```
<FORM>
<INPUT TYPE="button" VALUE="Нажми здесь!" onClick="askuser()">
</FORM>
```

Здесь ничего нового, простая форма с кнопкой, которая запускает функцию `askuser()` (спросить пользователя) при нажатии кнопки.

Фрагмент сценария с функцией:

```
function askuser()
 {
  var answer="   "
  var statement="Отвечайте да или нет"
  var answer=prompt("Вы любите спорт?")
  if ( answer == "да")
```

```
 {statement="Я тоже люблю спорт!"}
if(answer == "нет")
 {statement="Я тоже ненавижу спорт!"}
alert(statement)
}
```

Значение переменной `answer` (ответ) равно трем пробелам. Это пустое пространство будет заполнено тем ответом, который пользователь введет в поле ввода.

На те случаи, когда пользователь не отвечает `"да"` или `"нет"`, создается переменная `statement`. Скоро будет понятно, зачем это нужно.

Затем переменная `answer` задается как результат запроса `prompt`. Теперь у нас две переменные под одним именем. Помните об этом.

Следом за `if` идет условие в (круглых скобках). Не забывайте о них.

В условии, которое состоит из того, что находится между скобок, используется не знак =, а знак == (два знака равенства)! Одинарный знак = используется вне скобок.

Помните, строковые значения должны заключаться в кавычки. `"да"` и `"нет"` являются строками текста.

Процесс происходит следующим образом:

1. `prompt` предлагает ответить на вопрос;
2. проверяется введенное значение;
3. если ответ `"да"`, появляется окно со словами: `"Я тоже люблю спорт!"`
4. если ответ `"нет"`, появляется окно со словами: `"Я тоже ненавижу спорт!"`
5. если ответ ни тот, ни другой, переменная `answer` остается пустой и строка `"Отвечайте да или нет"` посылается в `alert`.

Помните, что JavaScript различает регистр символов. То есть, если

написать "НЕТ" или "Нет", условие не будет выполнено! Чтобы условие было верно, необходимо ввести "нет". Исправить это можно, добавив еще несколько условий IF на все случаи жизни.

Задание

Перепишите программу так, чтобы она спрашивала пол пользователя. Пусть в зависимости от ответа меняется фоновый цвет страницы. Помните, что в JavaScript различаются строчные и заглавные буквы, так что будьте внимательны в своих условиях. Например:

```
if (answer == "М")
```

Если ввести "м", то условие не будет выполняться!

Возможное решение

```
<html>
  <head>
 <SCRIPT type="text/javascript">
  function askuser()
   {
    answer=prompt("Вы мужчина (М) или женщина (Ж)?")
    if ( answer == "Ж")
      {document.bgColor="pink"}
    if(answer == "М")
      {document.bgColor="lightblue"}
   }
</script>

</head>
<body bgcolor="lightyellow">
 <h1>Привет</h1>
 <form>
  <input type="button"
    value="Укажите свой пол"
    onClick="askuser()">
 </form>
```

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

Необходимо изменить текст в нескольких местах, но основное для получения результата состоит в том, что ввод пользователя используется для изменения `document.bgColor`.

Операторы if/else

Примеры работы с условиями.

Концепция

Этот второй пример с `if` включает случайное число, две функции и знакомство с `If / Else`.

Операторы `If / Else` (если/иначе) предоставляют дополнительный контроль над программой, позволяя принимать решение в обоих случаях: и когда условие выполнено, и когда не выполнено.

Сценарий

```
<HTML>
<HEAD>
<SCRIPT type="text/javascript">
 function rand()
  {now=new Date();
   num=(now.getSeconds())%10;
   num=num+1;
   }
 function guessnum()
  {guess=prompt("Угадай, какое?")
   if (eval(guess) == num)
   {alert("ПРАВИЛЬНО!!");
    rand();
    }
     else
    alert("Нет.  Попробуй еще.");
      }
  </SCRIPT>
</HEAD>
<BODY onLoad="rand()">

  <h1>Я загадал число от 1 до 10</h1>
```

```
  <FORM NAME="myform">

    <INPUT TYPE="button" VALUE="Угадай"
      NAME="b1" onClick="guessnum()">

  </FORM>
</BODY>
</HTML>
```

Результат работы сценария

На странице выводится текст "Я загадал число от 1 до 10" и кнопка с предложением "Угадай". При нажатии на кнопку выводится окно с полем ввода и словами "Угадай, какое". В зависимости от введенного числа пользователь получит то или иное сообщение.

Разбор сценария

Начнем с команды BODY.

```
<body bgcolor="white" onLoad="rand()">
```

На этот раз функция запускается не событием onClick в описании кнопки, а событием onLoad в команде BODY. В этом случае к тому времени, когда пользователь нажмет кнопку, число уже будет выбрано. Если сделать это по-другому, то каждый раз, нажимая на кнопку, вы будете получать новое случайное число. А оно должно оставаться одним и тем же, пока вы пытаетесь угадать.

Теперь первая функция:

```
function rand() {
now=new Date();
num=(now.getSeconds())%10;
num=num+1;
}
```

Функция выбирает наугад число из даты и времени от 0 до 9 и

присваивает его `num`. Потом прибавляет к `num` единицу, чтобы выбор осуществлялся между 1 и 10. Мы делали это в прошлом уроке.

Теперь вторая функция:

```
function guessnum()
 {guess=prompt("Угадай, какое?")
  if (eval(guess) == num)
  {alert("ПРАВИЛЬНО!!!");
   rand();
   }
    else
   alert("Нет. Попробуй еще раз.");
   }
```

В памяти компьютера уже есть число, полученное через первую функцию. Вторая дает возможность угадать его. Смотрите, что происходит:

1. С помощью запроса prompt создается переменная `guess` (догадка). Функция `eval()` вычисляет или выполняет строку в скобках (выражение, команду или последовательность команд) и подставляет полученное значение вместо себя. Она не является методом какого-либо объекта, но может использовать свойства уже существующего. Обратите внимание на {фигурные скобки}. Все это очень похоже на Урок 21.
2. Переходим к IF/Else. Если (`if`) `guess` (догадка) равна загаданному числу `num`, тогда запускается команда `alert("ПРАВИЛЬНО")`.
3. Если это не так (`else`), тогда запускается другая команда `alert`.

Остальное уже знакомо:

```
<form name="myform" action="">
<input type="button" value="Угадай" name="b1" onClick="guessnum()">
</form>
```

Кнопка запускает функцию, которая дает возможность угадать задуманное число.

Задание

Измените сценарий урока так, чтобы при неверной догадке он сообщал пользователю, что он назвал слишком большое или слишком маленькое число.

Подсказка! В этом случае возможны только три результата: слишком много, слишком мало или правильно. Подумайте вот о чем: нужна ли команда `Else` или будет достаточно и парочки дополнительных `If`?

Возможное решение

Используем подсказку. Возможны только три результата: слишком мало, слишком много и точно. То есть каждый раз, когда пользователь вводит свою догадку, будет задействовано одно из трех условий. Здесь даже не понадобится `ELSE`. Таким образом, требуется только три утверждения `IF`.

Обратите внимание на команды < и > в сценарии. В данном случае они означают то же самое, что и на уроках математики: больше и меньше.

```
<html>
<head>
 <SCRIPT type="text/javascript">
  function rand()
   {now=new Date();
    num=(now.getSeconds())%10;
    num=num+1;
   }
  function guessnum()
   {guess=prompt("Угадай, какое?");
   if (eval(guess) == num)
     {alert("Точно!!!");
     }
   if(eval(guess) > num)
    {alert("Слишком много, жми еще.")}
   if(eval(guess) < num)
    {alert("Слишком мало, жми еще.")}
```

```
    }
  </script>
 <body bgcolor="white" onLoad="rand()">
  <h2>Я загадал число от 1 до 10</h2>
  <form name="myform">
   <input type="button" value="Угадай" name="b1" onClick="guessnum()">
  </form>
 </body>
 </html>
```

Случайный выбор фраз и изображений

Выбор и обработка случайных данных.

Концепция

Прежде чем мы перейдем к следующей теме, давайте рассмотрим еще один пример использования `IF`. Важно, чтобы вы хорошо овладели этой техникой ветвления, которая позволяет создавать более оригинальные и более интерактивные программы.

Сценарий

```
<HTML>
<BODY>
<SCRIPT type="text/javascript">
  var0="От пирогов не толстеют"
  var1="Кто ходит в гости по утрам"
  var2="До пятницы я совершенно свободен"
 now=new Date()
 num=(now.getSeconds() )%3

  if (num == 0)
   {cliche=var0}
  if (num == 1)
   {cliche=var1}
  if (num == 2)
    {cliche=var2}
 document.write(cliche + "<br>")
 document.write("Случайное число: "
  + num)
</SCRIPT>
<p>.... как я обычно говорю.
</BODY>
</HTML>
```

Результат работы сценария

Если страницу с этим сценарием перезагрузить несколько раз, то случайным образом будет выводиться одна из фраз и случайное число `0`, `1` или `2`.

Разбор сценария

Начнем со случайного числа:

```
var0="От пирогов не толстеют"
  var1="Кто ходит в гости по утрам"
  var2="До пятницы я совершенно свободен"
 now=new Date()
 num=(now.getSeconds() )%3
document.write("Случайное число: "
   + num)
```

Оператор `document.write` должен располагаться на одной строке!

Это вы можете разобрать и сами, все было на прошлых уроках.

Мы создали три переменные. Это неизменяемые фрагменты текста и потому заключены в двойные кавычки.

Следующий шаг: программа создает случайное число с помощью часов компьютера. `%3` указывает на то, что будет выбрана цифра из 0, 1 и 2. На этот раз мы не прибавляем к num единицу, так как нам подходит и 0.

Наконец, команда `document.write()` используется для вывода выбранного числа на странице.

Теперь посмотрим на вторую часть сценария:

```
if (num == 0)
   {cliche=var0}
  if (num == 1)
   {cliche=var1}
  if (num == 2)
    {cliche=var2}
 document.write(cliche + "<br>") >
```

Помните, что условия после `IF` требуют двойного знака равенства `==`.

Если условие верно, будет выполнена команда, заключенная в {фигурные скобки}. Возможны только три результата, поэтому мы написали три условия, чтобы одно из них оказалось верным.

Обратите внимание еще раз, что условие заключено в (круглые скобки), а результат — в {фигурные}.

Наконец команда `document.write(cliche)` выведет на странице то изречение, которое было присвоено переменной `cliche` (см. начало сценария).

Задание

Измените программу так, чтобы она показывала рисунок, выбранный случайным образом из трех: pic1.gif, pic2.gif и pic3.gif.

Возможное решение

Необходимо создать три переменные для трех рисунков и вписать имена переменных в команду `document.write`. Ниже представлены два варианта решения. Второй немного более компактный и немного более хитроумный. Попробуйте разобраться в нем самостоятельно.

Первый вариант

```
<body bgcolor="xxxxxx">
<center>
<h1>Моя домашняя страница</h1>
<script language="JavaScript">
  var1="pic1.gif"
  var2="pic2.gif";
  var3="pic3.gif"
  now=new Date()
  num=(now.getSeconds() )%3
  num=num+1
  quot="'"
```

```
    document.write("Случайная цифра: " + num + "<br>")
    if (num == 1)
       {cliche=var1}
    if (num == 2)
       {cliche=var2}
    if (num == 3)
       {cliche=var3}
   document.write("<img src=" + quot + cliche + quot +">")
  </script>
  <p>Такое у меня сегодня настроение.
  </center>
  </body>
  </html>
```

Для самых умных

```
  <h1>Моя домашняя страница</h1>
  <script language="JavaScript">
    var1="pic1.gif"
    var2="pic2.gif";
    var3="pic3.gif"
    now=new Date()
    num=(now.getSeconds() )%3
    num=num+1
    quot="'"
    document.write("Случайная цифра: " + num + "<br>")
    document.write("<img src=" + quot + "pic" + eval(num) + ".gif" + quot +">")
  </script>
  <p>Такое у меня сегодня настроение.
  </center>
  </body>
  </html>
```

Введение в циклы for

Повторяющиеся действия. Циклы с предусловием.

Концепция

Во всех языках программирования имеются средства организации ветвления. В JavaScript для этого используется оператор `IF`, который мы только что рассмотрели.

Во всех языках программирования имеются также средства организации повторяющихся операций или циклов. В JavaScript бывают циклы двух видов: `While` и `For`.

Обычно циклы `For` используются, когда известно количество повторений, а циклы `While` — когда точно не известно, сколько раз нужно повторить цикл. В данном примере будет рассмотрен цикл `For`.

Сценарий

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<H3>Посчитаем от одного до пяти:</H3>
<SCRIPT type="text/javascript">
for (i=1; i<=5; i=i+1)
{
document.write(i + "<BR>");
}
</SCRIPT>
</BODY>
</HTML>
```

...вот и все.

Результат работы сценария

На странице столбиком выводятся числа от 1 до 5.

```
1
2
3
4
5
```

Разбор сценария

Во-первых, какой короткий сценарий! Вот чистый сценарий без текста, который добавлен только для внешнего оформления:

```
<SCRIPT type="text/javascript">
  for (i=1; i<=5; i=i+1)
{
document.write(i + "<BR>");
}
 </script>
```

Посмотрим на синтаксис оператора `for`, `for (i=1; i<=5; i=i+1)`. В нем три части. Все они разделены точкой с запятой. Рассмотрим их по порядку.

`i=1` задает начальное значение переменной, управляющей циклом. В данном случае это `1`, но можно было бы задать `10` или `100`. Это просто начальная точка отсчета цикла.

`i<=5` — условие, определяющее, сколько в цикле будет повторений. В нашем случае цикл будет повторяться до тех пор, пока `i` не станет больше пяти. Видите? Мы начали с одного и досчитаем до пяти.

`i=i+1` определяет значение шага цикла. В нашем случае программа будет прибавлять `1` к `i` в конце цикла. Программа может прибавлять и `2`, и `20`, и `567`. Нам просто надо увеличивать `i` на `1`.

Наконец оператор `document.write`, выводит число на страницу. Обратите внимание на `<BR>` в операторе `document.write` — это заставляет каждое число выводиться в новой строке. С таким же

успехом можно было вывести их в одну строку, разделив запятыми и просто изменяя часть текста, которая появляется после каждого числа.

Этот код JavaScript повторится пять раз и выведет на странице цифры от `1` до `5`. Мы могли бы заставить его досчитать до миллиона, но это потребовало бы слишком много пространства Web-страницы.

Задание

Напишите документ HTML с текстом "Сюрприз!", заключенным в теги `<H2>` и расположенным на самом верху страницы.. Начните с белого фона. Потом с помощью JavaScript досчитайте до `50000`.

В этот момент цвет фона меняется на желтый и появляется текст: "Скоро будет еще один цветной сюрприз..."

Снова досчитайте до 50 тысяч, и тогда фон должен опять поменяться. Успеха.

Подсказка: никаких команд в {фигурные скобки} ставить не надо.

Возможное решение

Можно было просто дважды скопировать сценарий из урока, поменяв `5` на 50 тысяч. Это легкая часть работы. Более трудная состоит в реализации события. Мы удалили `document.write` после цикла и поставили вместо этого `document.bgColor=`.

```
<html>
<head>
</head>
<body bgcolor="white">
<h2>Сюрприз!</h2>
<SCRIPT type="text/javascript">
  for (i=1; i<=50000; i=i+1)
  {
  }
  document.bgColor="yellow"
```

```
</script>
И еще один сюрприз...
<script language="JavaScript">
  for (i=1; i<=50000; i=i+1)
  {
  }
  document.bgColor="pink"
</script>
</body>
</html>
```

Введение в циклы while

Повторяющиеся действия. Циклы с постусловием.

Концепция

В этом примере рассматривается цикл `While`. Помните, мы говорили, что циклы `For` используются, когда известно, сколько раз нужно их повторять, а циклы `While` — когда не известно. Этот пример нарушит правило! Он покажет, как пользоваться переменными, чтобы сосчитать количество повторений цикла и подготовиться к заданию.

Сценарий

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT type="text/javascript">
 loops=3
 num=1
 while (num <= loops)
   {
   document.write("Добро ")
   num=num+1
   }
   document.write("Пожаловать!")
  </SCRIPT>
</BODY>
</HTML>
```

Результат работы сценария

На странице выводится фраза:

Добро Добро Добро Пожаловать

Разбор сценария

```
<SCRIPT type="text/javascript">
        loops=3
        num=1
       while (num <= loops)
        {
           document.write("Добро ")
           num=num+1
         }
       document.write("Пожаловать!")
    </SCRIPT>
```

Еще одна коротенькая программа, мы ее быстро разберем.

Синтаксис операторов `While` и `For` похож. Разница в том, что мы задаем начальное значение индекса и шаг изменения прямо в команде `For`. Команда `While` содержит только условие.

`while(num<=loops)` говорит программе, что она должна повторять цикл, пока значение `num` меньше или равно значению переменной `loops`. Видите знак <=? Другими словами, программа повторит цикл трижды, один раз для `num=1`, один для `num=2` и еще один для `num=3`.

Каждый раз, когда программа выполняет цикл, она пишет `"Добро"` и прибавляет `1` к `num`. Когда num доходит до 4, цикл заканчивается. 4 больше `3`, поэтому программа напишет `"Добро"` три раза.

Завершающий штрих — команда `document.write` со словом `"Пожаловать!"`.

Задание

Первое: заставьте работать программу этого урока. Потом измените ее так, чтобы пользователь получал запрос: `"Сколько раз пожелать Вам Добро пожаловать?"` Для ответа создайте переменную. Помните команду `eval()`, которая преобразует текст в число? Затем программа должна написать `"Добро"` столько раз, сколько ее просили.

Возможное решение

Создается переменная с именем `tellme`, которой присваивается результат запроса `prompt`. Затем эта переменная с помощью функции `eval()` преобразуется в число `loops`.

Дальше все как в сценарии урока:

```
<html>
<head>
</head>
<body bgcolor="white">
<SCRIPT type="text/javascript">
 tellme=prompt("Сколько раз пожелать Вам Добро пожаловать?")
 loops=eval(tellme)
 num=1
 while (num <= loops)
   {
   document.write("Добро ")
   num=num+1
   }
   document.write("Пожаловать!")
</script>
<p>
THE END
</body>
</html>
```

Массивы

Введение в массивы. Их создание и обработка.

Концепция

В этом уроке рассматриваются массивы (array). Мы уже изучили переменные. Каждая переменная содержит в данный момент одно значение, но иногда требуется использовать массив (array) или переменную, которая содержит множество значений.

В примере урока программа предлагает пользователю угадать телевизионный канал из перечня телеканалов. Запрос повторяется до тех пор, пока пользователь не угадает. Каждый раз при нажатии кнопки случайным образом выбирается новый телеканал.

Сценарий

```
<HTML>
<HEAD>
<SCRIPT type="text/javascript">
    tv=new Array()
    tv[0]="ОРТ"
    tv[1]="РТР"
    tv[2]="ТВЦ"
    tv[3]="НТВ"
    tv[4]="ТВ6"
    num=0
function picktv()
   {
     now=new Date()
     num=(now.getSeconds())%5
    }
function whichtv()
{
picktv()
guess=" "
while (tv[num] != guess.toUpperCase())
```

```
{guess=prompt("Угадайте мой любимый телеканал:
 ОРТ, РТР, ТВЦ, НТВ или ТВ6?")

 if (guess.toUpperCase() == tv[num])
 {alert("Это мой любимый телеканал!")}
 else
 {alert("Нет, попробуйте еще раз.")}}
 }
</SCRIPT>
</HEAD>
<BODY>
<FORM>

<INPUT TYPE="button" VALUE="Угадайте телеканал!"
    onClick="whichtv()">

</FORM>
</BODY>
</HTML>
```

Результат работы сценария

На странице выводится кнопка с надписью `"Угадайте телеканал!"`, при нажатии на которую выводится окно с предложением угадать любимый канал из перечисленных.

В зависимости от ввода пользователю предлагается попробовать еще раз или программа завершается.

Разбор сценария

```
<SCRIPT type="text/javascript">
    tv=new Array()
    tv[0]="ОРТ"
    tv[1]="РТР"
    tv[2]="ТВЦ"
    tv[3]="НТВ"
```

```
tv[4]="ТВ6"
num=0
```

`tv=new Array()` объявляет `tv` как массив объектов. С пустыми (скобками) массив может быть какой угодно длины. Можно также указать длину массива, например, `tv=new Array(5)`.

Помните, что массив может иметь множество значений. Можно представить себе массив в виде таблицы:

tv				
tv[0]	tv[1]	tv[2]	tv[3]	tv[4]
ОРТ	РТР	ТВЦ	НТВ	ТВ6

Обратите внимание, мы заранее указываем переменную `num`, у которой одно значение, равное `0`, и массив `tv`, который имеет 5 значений.

Теперь функция `picktv()`:

```
function picktv()
   {
     now=new Date()
     num=(now.getSeconds())%5
    }
```

Функция `picktv()` случайно выбирает число от 0 до 4, которое становится индексом `tv`. Помните, от нуля до четырех ПЯТЬ чисел. То есть если `num` равно 2, то любимый телеканал — `tv[2]`, или ТВЦ.

Теперь функция `whichtv()`:

```
function whichtv()
{
picktv()
guess=" "
while (tv[num] != guess.toUpperCase())

{guess=prompt("Угадайте мой любимый телеканал:
  ОРТ, РТР, ТВЦ, НТВ или ТВ6?")
```

```
if (guess.toUpperCase() == tv[num])
{alert("Это мой любимый телеканал!")}
else
{alert("Нет, попробуйте еще раз.")}}
}
```

Команда `guess=prompt(...)` должна находиться полностью на одной строке.

Вот кое-что новое! Видите, первым делом функция вызывает другую функцию, `picktv()`. Таким образом, когда бы вы ни нажали на кнопку, будет выбираться новый телеканал.

Обратите внимание на строку `while (tv[num] != guess.toUpperCase())`. Метод или действие `toUpperCase()` (в верхний регистр) используется для перевода всего, что было введено, в верхний регистр символов.

Программа повторяет цикл `While`, пока пользователь не угадает правильный телеканал. Фрагмент с циклом `While` должен быть уже вполне знакомым.

Обратите внимание на операторы `If` и `Else`. В этой игре возможны только два результата: либо вы правы, либо ошибаетесь.

Теперь кнопка, которая все это запускает:

```
<FORM>
<INPUT TYPE="button" VALUE="Угадай телеканал!"
   onClick="whichtv()">
</FORM>
```

Тут ничего нового.

Еще кое-что о массивах

В JavaScript есть несколько встроенных массивов. Формы можно хранить в массивах. На форму можно ссылаться с помощью

`document.myform` или `document.forms[0]`, если это первая форма. Массивы всегда начинаются с нуля. Вторая форма будет `document.forms[1]`. Третья — `document.forms[2]` и так далее...

Изображения также можно хранить во встроенном массиве. Можно ссылаться на pic1.gif как `document.pic1.src` или как `document.images[0].src`. Просто продолжайте следовать схеме, указывая номер в [квадратных скобках].

Теперь, наверное, стало понятно, что такое массивы.

Задание

Напишите программу JavaScript, которая содержит кнопку с надписью: "Щелкните, чтобы попасть на случайный сайт". Когда пользователь нажимает ее, выполняется функция, которая выберет случайное число и сайт из массива с помощью команды JavaScript `top.location.href = url[num]`. `top` (вершина) — это свойство объекта `window`, оно относится к главному окну браузера. `location.href`, другой объект и свойство, содержит адрес URL.

Возможное решение

Это задание очень напоминает пример из этого урока, за исключением того, что нужно указать ряд адресов URL по схеме `url[0]`. Однако в результате сценарий должен отправить пользователя на выбранную страницу.

```
<html>
<head>
<script language="JavaScript">
    url=new Array()
    url[0]="http://www.jsp.newmail.ru/les5.htm"
    url[1]="http://www.jsp.newmail.ru/les10.htm"
    url[2]="http://www.jsp.newmail.ru/les15.htm"
    url[3]="http://www.jsp.newmail.ru/les20.htm"
```

```
function rand()
  {
   now=new Date()
   num=(now.getSeconds())%4
   top.location.href = url[num]
  }
</script>
</head>
<body>
<center>
  <h2>Случайный URL</h2>
  <form>
    <input type="button" value="Случайное блуждание по сайтам!" onClicl
  </form>
</center>
</body>
</html>
```

Слайд-шоу

Динамическая загрузка и обновление изображений.

Концепция

Последние три урока посвящены тому, чтобы помочь читателю собрать в целое полученные знания. Далее следуют три популярных сценария JavaScript, которые будут частично проанализированы. Часть работы будет поручена читателю.

Посмотрите на скрипт и попробуйте разобраться, как и что он делает. Выделите части, которые создают определенные события. А лучше всего постарайтесь немного изменить его в лучшую сторону.

В этом примере показано слайд-шоу. Пользователь щелкает по ссылке и переходит к следующей картинке. Мы воспользуемся командой `If` и переменной `num`. Ничего нового? Не совсем!

Сценарий

```
<HTML>
<HEAD>
<SCRIPT type="text/javascript">
  var num=1
  img1 = new Image ()
  img1.src = "pic1.gif"
  img2 = new Image ()
  img2.src = "pic2.gif"
  img3 = new Image ()
  img3.src = "pic3.gif"
function slideshow()
  {
   num=num+1
   if (num==4)
    {num=1}
document.animal.src=eval("img"+num+".src")
   }
```

```
</SCRIPT>
</HEAD>
<BODY>
<CENTER>
<IMG SRC="pic1.gif" NAME="animal" BORDER=0>
<p>

<A HREF="JavaScript:slideshow()">
  Щелкните, чтобы увидеть следующую картинку</A>

</CENTER>
</BODY>
</HTML>
```

Результат работы сценария

На странице выводится изображение pic1.gif и ссылка "Щелкните, чтобы увидеть следующую картинку", при щелчке на которой изображение изменяется на pic2.gif и так далее.

Разбор сценария

Разделим его на две части:

```
<SCRIPT type="text/javascript">
        var num=1
        img1 = new Image ()
        img1.src = "pic1.gif"
        img2 = new Image ()
        img2.src = "pic2.gif"
        img3 = new Image ()
        img3.src = "pic3.gif"
```

Новый момент! `num=1` не находится внутри функции. Да и вообще ни одна команда не находится внутри функции. `num` — это переменная. Объявляя переменную в начале сценария JavaScript вне функции, мы делаем ее глобальной переменной, то есть доступной для любой следующей далее функции.

`img1 = new Image()` объявляет новый объект image (изображение). `img1.src=` источник объекта, определяет, какая картинка хранится в объекте image. Это стандартная схема. Она должна использоваться, когда задается ряд изображений, как в этом сценарии. pic1.gif хранится в `img1.src`. pic2.gif хранится в `img2.src`.

(В круглых скобках) может содержаться ширина и высота каждого рисунка. Это не обязательно, но желательно, так как ускоряет время загрузки.

Объекты image доступны любой следующей далее функции. Так же, как и переменная `num`, они находятся вне функции. Они просто перечислены здесь, чтобы их могла использовать любая часть программы. Помещая все объекты изображений вне функции, программа загружает изображения. Это будет еще более важно в следующем примере анимации. Во время анимации пользователь не захочет ждать загрузки каждой картинки с сервера. Поэтому требуется предварительная загрузка.

Теперь вторая часть:

```
function slideshow()
{
num=num+1
if (num==4)
  {num=1}
document.animal.src=eval("img"+num+".src")
}
</script>
</head>
<body>
```

Начальное значение `num` равно `1`. Это было задано в первом фрагменте. Когда пользователь щелкает на тексте ссылки, запускается функция `slideshow`.

`slideshow()` прибавляет к `num` единицу. Когда num доходит до `4`, то `num` снова присваивается значение `1`. `document.animal.src` меняется на `img` плюс значение `num` и плюс `.src`. Например, если

`num = 1`, то `document.animal.src` становится img1.src.

Обратите внимание, что команда `eval()` преобразует "img1.src" в указание на источник изображения. Без нее это был бы простой набор букв.

И, наконец, команда, которая все это запускает в работу :

```
<a href="JavaScript:slideshow()">
Щелкните, чтобы увидеть следующую картинку</a>
```

Тут кое-что новое. Видите, вызывается не функция, а функция с префиксом JavaScript? Это позволяет использовать все части сценария, а не только функцию. Если написать здесь только функцию, то у вас не будет рисунков, потому что они останутся за скобками.

Задание

Перепишите приведенный выше сценарий JavaScript. Покажите первым pic1.gif, как в примере. Однако дальше новый сценарий должен показать img3.src (`num=3`), потом img2.src, потом img1.src. Когда `num=0`, измените `num` на `3`.

В общем, перепишите сценарий "задом наперед".

Возможное решение

Если `+` означает сложение, то `-` означает вычитание. Посмотрите на сценарий. Менять местами рисунки не надо, хотя и можно. Просто сделайте так, чтобы из `num` каждый раз вычиталась единица, а когда num добирается до нуля, снова начинайте с трех.

Вот сценарий:

```
<html>
<head>
<SCRIPT type="text/javascript">
   num=4
   img1 = new Image (); img1.src = "pic1.gif"
```

```
    img2 = new Image (); img2.src = "pic2.gif"
    img3 = new Image (); img3.src = "pic3.gif"
    function slideshow()
    {
      num=num-1
      if (num==0)
        {num=3}
     document.mypic.src=eval("img"+num+".src")
    }
</script>
</head>
<body>
<center>
<img src="pic1.gif" name="mypic" border=0>
<p>
<a href="JavaScript:slideshow()">
Щелкните, чтобы увидеть следующую картинку</a>
</center>
</body>
</html>
```

Анимация

Основы мультипликации в JavaScript.

Концепция

Здесь с помощью JavaScript создается анимационная картинка. Для анимации особенно важно, чтобы все изображения были заранее загружены в память компьютера. Помните, как это делается? С помощью команд `new Image()` и img.src, стоящих вне функции.

Если не позаботиться об этом заранее, то пользователю придется ждать, пока каждая картинка будет загружаться с сервера. Какая же это будет анимация!

Сценарий

```
<HTML>
<HEAD>
<SCRIPT type="text/javascript">
 var num=1
 img1 = new Image (150,150)
 img1.src = "pic1.gif"
 img2 = new Image (150,150)
 img2.src = "pic2.gif"
 img3 = new Image (150,150)
 img3.src = "pic3.gif"
  function startshow()
  {
   for (i=0; i<21; i=i+1)
{document.mypic.src=eval("img"+num+".src")
    for(x=1; x<8000; x=x+1)
  {}
    num=num+1
    if(num==4)
    {num=1}
   }
   document.mypic.src=img1.src
```

```
    }
  </SCRIPT>
  </HEAD>
  <BODY>
  <CENTER>
  <IMG SRC="pic1.gif" NAME="mypic" BORDER=0 alt="">
  <p>

  <A HREF="JavaScript:startshow()">
    Показать анимацию</a>

  </CENTER>
  </BODY>
  </HTML>
```

Результат работы сценария

На странице выводится изображение pic1.gif и текстовая ссылка " Показать анимацию", при нажатии на которую начинается последовательная смена изображений (анимация).

(Сценарий может не работать в браузере MSIE.)

Разбор сценария

Мы начинаем с предварительной загрузки изображений. Обратите внимание, что это происходит вне функции, так же, как и задание переменной `num`. Это выглядит следующим образом:

```
  <SCRIPT type="text/javascript">
     var num=1
     img1 = new Image (150,150)
     img1.src = "pic1.gif"
     img2 = new Image (150,150)
     img2.src = "pic2.gif"
     img3 = new Image (150,150)
     img3.src = "pic3.gif"
```

Теперь функция:

```
function startshow()
 {
  for (i=0; i<21; i=i+1)
{document.mypic.src=eval("img"+num+".src")
    for(x=1; x<8000; x=x+1)
    {}
    num=num+1
    if(num==4)
     {num=1}
    }
  document.mypic.src=img1.src
    }
</SCRIPT>
```

`startshow()` содержит новый элемент, вложенные циклы! Видите, внутри первого цикла `for` находится второй (слово `for` повторяется дважды.) Второй цикл замедляет смену картинок, чтобы пользователь смог их разглядеть.

Обратите внимание, в {фигурных скобках} цикла ничего нет. Вложенный цикл считает от `1 до 8000` после появления каждой картинки. На это уходят доли секунды.

Внешний цикл заставляет программу повторяться в цикле `21` раз. Видите это в функции: в первом операторе цикла `for (i=0; i<21; i=i+1)`. Так как анимация состоит из трех картинок, она будет повторена 7 раз: 7 X 3 = 21.

Когда цикл закончится, картинка снова вернется к pic1.gif.

Вот команда, которая помещает первое изображение на страницу:

```
<IMG SRC="pic1.gif" NAME="mypic" BORDER=0 alt="">
```

Обратите внимание, что имя задано как `"mypic"`. Если посмотреть на иерархию операторов в сценарии, то оно находится между `document` и `src`.

И наконец ссылка, которая запускает анимацию:

```
<A HREF="JavaScript:startshow()">Показать анимацию</a>
```

Снова команда `HREF` указывает на `JavaScript:функция()`, чтобы все глобальные переменные были доступны.

Задание

Перепишите программу анимации. Пусть первым будет pic1.gif, как в данном примере, но вставьте его в форму. Включите в нее текстовое поле, куда пользователь мог бы ввести слово slow (медленно), medium (умеренно) или fast (быстро), выбирая скорость смены картинок. Пусть medium стоит по умолчанию. `800` будет "быстро", `1600` — "умеренно", `2400` — "медленно". Запускать анимацию должна текстовая ссылка "Показать анимацию".

Возможное решение

В программу включено окно `alert`, чтобы показывать значение переменной `howfast` (как быстро). `alert` — это хороший инструмент отладки программ. Просмотр значений переменных часто помогает определить, где программа ведет себя не так, как хотелось бы.

Еще обратите особое внимание на параметр `sp` (speed, скорость) внутри функции `startshow()`. Он позволяет передавать в функцию данные, которые вводит пользователь.

И последнее — метод `toUpperCase()`. С его помощью все варианты написания `"slow"`, `"Slow"`, `"sLOW"` преобразуются в `"SLOW"`. Хороший прием, не правда ли?

Вот сценарий:

```
<html>
<head>
<SCRIPT type="text/javascript">
   num=1
```

```
    img1 = new Image ()
    img1.src = "pic1.gif"
    img2 = new Image ()
    img2.src = "pic2.gif"
    img3 = new Image ()
    img3.src = "pic3.gif"
    function startshow(sp)
    {
      howfast=1600
      if (sp.toUpperCase() == "SLOW")
        {howfast=2400}
      if(sp.toUpperCase() == "FAST")
        {howfast=800}
      alert(howfast)
      for (i=0; i<21; i=i+1)
        {document.mypic.src=eval("img"+num+".src")
         for(x=1; x<howfast; x=x+1)
           {}
         num=num+1
         if(num==4)
           {num=1}
         }
         document.mypic.src=img1.src
    }
</script>
</head>
<body>
<center>
<form name="myform">
<img src="pic1.gif" name="mypic" border=0>
<p>
Задайте скорость анимации: (fast, medium, slow) <input type="text" value="n
<p>
<a href="JavaScript:startshow(document.myform.speed.value)">Показать аним
</form>
</center>
</body>
</html>
```

Проверка данных в форме

Предварительная обработка данных форм.

Концепция

Наш последний сценарий JavaScript не из легких, можете в этом не сомневаться.

Снова придется вернуться к формам. В сегодняшнем примере JavaScript применяется для проверки данных, которые ввел пользователь. Нужно будет ввести в форму свое имя и номер телефона из 7 или 9 знаков (xxxxxxx или xxx-xx-xx). Необходимо также проверить, что первые 3 символа являются цифрами. Это немного сложнее, чем все остальное, но с этим необходимо разобраться. Проверка данных часто является важной задачей для программистов.

Этот пример возвращает нас к свойству `length` (длина) и показывает в действии два новых метода: `indexOf()`, `charAt()`. Сам сценарий будет длиннее, чем обычно.

Сценарий

```
<HTML>
<HEAD>
<SCRIPT type="text/javascript">
function validfn(fnm)
{
fnlen=fnm.length
if (fnlen == 0)
{alert("Вы должны ввести свое имя")
document.dataentry.fn.focus()}
}
function validphone(phone)
{
len=phone.length
digits="0123456789"
if(len != 7 && len != 9)
```

```
{alert("Неверное количество знаков в номере")
document.dataentry.phone.focus()}

for(i=0; i<3; i++)
{if (digits.indexOf(phone.charAt(i))<0)
{alert("Это должны быть цифры")
document.dataentry.phone.focus()
break}
}

}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="dataentry">
<h2>Подтверждение данных</h2>

Введите свое имя:<br>
  <INPUT TYPE="text" NAME="fn"
    onBlur="validfn(fn.value)">

<SCRIPT LANGUAGE="JavaScript">
  document.dataentry.fn.focus()
</SCRIPT>

Введите номер телефона (xxx-xx-xx):<br>
    <INPUT TYPE="text" NAME="phone" SIZE=10 >

<INPUT TYPE="button"  VALUE="Отправить"
     onClick="validphone(phone.value)">

</BODY>
</HTML>
```

Результат работы сценария

На странице выводятся два поля ввода с просьбой ввести свое имя и номер телефона. В случае недопустимого ввода выдается сообщение об

ошибке.

Разбор сценария

Сценарий достаточно простой. Имеется две функции, `validfn()` и `validphone()`. Одна проверяет, введено ли имя, другая проверяет телефонный номер. Займемся первой:

```
function validfn(fnm)
{
  fnlen=fnm.length
  if (fnlen == 0)
  {alert("Вы должны ввести свое имя")
  document.dataentry.fn.focus()}
  }

<body>

<INPUT TYPE="text" NAME="fn"
   onBlur="validfn(fn.value)">
```

Функция `validfn(fnm)` вызывается обработчиком события `onBlur`. `onBlur` запускается, когда курсор переходит на следующий элемент, в данном случае, когда мы выходим из текстового поля `fn`. Этот обработчик рассматривался в пятом уроке.

Заметьте, что параметр `fn.value` передается из формы в функцию, где получает новое имя `fnm`. `fn.value` можно было бы обозначить как `document.dataentry.fn.value`, но раз мы находимся в документе и внутри формы, это не обязательно.

Помните, содержимое поля формы передает команда `имя_формы.value`. Одного имени мало.

Переменной с именем `fnlen` присвоено значение длины имени пользователя. Таким образом, если пользователь введет имя "Коля", значение `fnlen` будет равно 4.

Если пользователь не вписал свое имя, значит, длина равна 0. Тогда

программа выводит окно с сообщением об ошибке и ставит курсор или фокус в поле для ввода имени. Форма не столько проверяет, правильно ли вписано имя, сколько было ли что-нибудь вписано вообще.

Теперь посмотрим, как программа проверяет телефонный номер:

```
function validphone(phone)
{
len=phone.length
digits="0123456789"
if(len != 7 && len != 9)
{alert("Неверное количество знаков в номере")
document.dataentry.phone.focus()}

for(i=0; i<3; i++)
{if (digits.indexOf(phone.charAt(i))<0)
{alert("Это должны быть цифры")
document.dataentry.phone.focus()
break}
}
```

Эта функция более длинная. Давайте разберем ее шаг за шагом. Во-первых, длина телефонного номера присваивается переменной `len`. Переменная `digits` содержит все десятичные цифры.

Потом оператор `If` проверяет, равна ли длина номера 7 или 9 знакам, хотя и звучит это несколько неуклюже. Двойной знак `&&` в Javascript означает "проверить оба свойства".

Если условие не выполнено, программа говорит пользователю о том, что он ввел неверное количество цифр, и снова устанавливает курсор или фокус в поле для ввода номера.

`for(i=0; i<3; i++)` проверяет первые `3` цифры номера одну за другой.

Выражение `if (digits.indexOf(phone.charAt(i))<0)` знакомит нас с двумя новыми методами: `indexOf()` и `charAt()`. Посмотрим на `phone.charAt(i)`. Предположим, телефонный номер

123, и `i = 2`. Знак на второй позиции — цифра 3. Это не опечатка! Помните, порядковые номера начинаем считать с нуля. Таким образом, `phone.charAt(0) = 1`, `phone.charAt(1) = 2`, а `phone.charAt(2) = 3`!

`indexOf` — это метод, дающий порядковый номер для заданного значения. С помощью `if (digits.indexOf(phone.charAt(i))<0)`, JavaScript ищет значение `phone.charAt(i)` в переменной `digits`.

Если телефонный номер 1234567 и `i = 1`, то программа ищет вторую цифру в переменной `digits` и находит ее, возвращая значение 1, так как `digits = "0123456789"`.

Если номер телефона 12д и `i = 2`, программа ищет `"д"` в переменной `digits`. Не найдя ее, она возвращает `-1`. Если значение `= -1` (`<0`), тогда появляется окно с сообщением об ошибке и курсор или фокус устанавливается на прежнее место. Для телефонного номера xxxxxxx так можно проверить все `7` цифр.

И последнее — код HTML для формы:

```
Введите свое имя:<br>
  <INPUT TYPE="text" NAME="fn"
    onBlur="validfn(fn.value)">

<SCRIPT LANGUAGE="JavaScript">
  document.dataentry.fn.focus()
</SCRIPT>

Введите номер телефона (xxx-xx-xx):<br>
    <INPUT TYPE="text" NAME="phone" SIZE=10>

<INPUT TYPE="button"  VALUE="Отправить"
     onClick="validphone(phone.value)">
```

При использовании JavaScript с формами давайте каждому элементу уникальное имя, которое будет связывать его с разделом JavaScript, который его обрабатывает. Такое связывание мы уже использовали

ранее. Просмотрите элементы формы и затем код JavaScript и определите, где одно связано с другим.

Задание

Во-первых, изучите сегодняшнюю программу и заставьте ее работать. Потом внесите несколько изменений. Попросите ввести телефонный номер в формате xxx-xxxx. Пусть функция `validphone(phone)` проверит, стоит ли дефис на позиции 3. Команда `!=` в JavaScript означает "не равно". Это может понадобиться.

Возможное решение

```
<html>
<head>

<SCRIPT type="text/javascript">
function validfn(fnm)
{
fnlen=fnm.length
if (fnlen == 0)
{alert("Необходимо ввести имя")
document.dataentry.fn.focus()}
}
function validphone(phone)
{
len=phone.length
digits="0123456789"
if(len != 8)
{alert("Неверное количество цифр в номере")
document.dataentry.phone.focus()}

if(len>3 && phone.charAt(3) != "-")
{alert("Четвертый знак должен быть '-'")
document.dataentry.phone.focus()}

for(i=0; i<3; i++)
{if (digits.indexOf(phone.charAt(i))<0)
```

```
{alert("Это должны быть цифры")
document.dataentry.phone.focus()
break}
}
}
</script>

</head>
<body>

<form name="dataentry">
<h3>Подтверждение ввода данных через JavaScript</h3>
<p>Введите имя:<br>
<input type="text" name="fn" onBlur="validfn(fn.value)" size="20"> <script
language="JavaScript">
document.dataentry.fn.focus()
</script></p>
<p>Введите номер телефона (xxx-xxxx):<br>
<input type="text" name="phone" size="10"></p>
<p><input type="button" value="Отправить"
onClick="validphone(phone.value)"></p>
</form>

</body>
</html>
```

Что дальше?

Обзор изученного материала.

В JavaScript есть и другие команды, но те, что показаны здесь, являются настоящими рабочими лошадками JavaScript. Рекомендуем купить какой-нибудь справочник по всем командам JavaScript. Если вы внимательно читали эти уроки и выполнили все задания, то вам не потребуется другой учебник.

В Интернет можно найти множество свободно доступных сценариев, которые можно использовать на своих страницах. Найдите, что вам интересно. Изучите. Попробуйте улучшить. У каждого учащегося наступает момент, когда он должен начать действовать самостоятельно.

Что мы изучили

- Массив (`Array`) имеет дело с переменными. Как правило, переменные имеют только одно значение. Однако массив является переменной с множеством значений.
- Запятая (,) используется для разделения нескольких обработчиков событий Javascript, когда они должны действовать одновременно.

- Комментарий является строкой текста, выделенной двойной косой чертой (`//`). Эта строка текста находится в сценарии, но не будет оказывать никакого влияния на его выполнение.

  Комментарий из нескольких строк можно создать с помощью команды `/*` в начале параграфа и команды `*/` в конце. Все, что находится между ними, будет считаться комментарием.

  Используйте команды комментария `<!--` и `-->` для комментирования текста Javascript, чтобы браузеры, которые не поддерживают JavaScript, просто игнорировали сценарий, не выводя текст. См. Урок 8.

- Ошибки

  Окно с сообщением об ошибке выводится, когда что-то в

сценарии не позволяет его выполнить. Ошибки бывают двух видов: ошибки времени выполнения (RunTime) и синтаксические ошибки (Syntax).

  - Ошибкой времени выполнения является ошибка, которая возникает, когда в коде JavaScript появляются неправильные выражения. Неправильная команда или несвоевременный формат будут порождать ошибку такого типа.
  - Синтаксическая ошибка создается, когда формат сценария или форма неправильны, найдена орфографическая ошибка, или текст не распознан. Синтаксические ошибки порождаются также в случае, когда команда начинается, но не заканчивается.

- Обработчик событий (Event Handler) является командой JavaScript, которая "встроена" в код HTML. Ее не нужно определять отдельно, как сам сценарий. Команда помещается в код HTML для создания взаимодействия между пользователем и страницей.

  - `onBlur` является обработчиком событий, который действует, когда активный элемент формы, такой как поле выбора, текст или текстовая область, становится неактивным. Другими словами, пользователь смещает фокус с элемента.
  - `onChange` является обработчиком событий, который действует, когда пользователь изменяет текст в элементе выбора, текста или текстовой области формы. Обычно эта команда используется для проверки ошибок при вводе пользователя.
  - `onClick` является обработчиком событий, который действует, когда пользователь щелкает на объекте, таком, как ссылка.
  - `onFocus` является обработчиком событий, который действует, когда пользователь выбирает элемент формы, такой, как поле выбора, текста или текстовую область.
  - `onLoad` является обработчиком событий, который запускает функцию, когда загружается страница. Команда помещается в теге `BODY` документа HTML.
  - `onMouseOut` является обработчиком событий, который

применяется в гипертекстовой ссылке для создания текста, когда когда указатель мыши смещается с элемента. Наиболее часто используется совместно с `onMouseOver`.

- `onMouseOver` является обработчиком событий, используемым в гипертекстовой ссылке, который реагирует, когда пользователь наводит указатель мыши на текст ссылки.
- `onSelect` является обработчиком событий, который активируется, когда пользователь выделяет текст в элементе формы текст или текстовая область.
- `onSubmit` является обработчиком событий, который активируется, когда пользователь щелкает на кнопке формы submit (Отправить).
- `onUnload` является обработчиком событий, который помещается в команде `BODY` документа HTML, и либо вызывает функцию, либо содержит команду для остановки выполнения функции, когда пользователь покидает страницу.

- Характеристики определяют свойства команды JavaScript. Наиболее обычным примером являются такие характеристики, как `toolbar`, `menubar`, `scrollbars`, `resizable`, `directories`, и `status`, которые используются для определения формы и вида нового окна.

  - `config` является командой, используемой в экземпляре объекта и метода window.open().Она определяет, что далее следуют характеристики нового окна.
  - `directories` является характеристикой, используемой в экземпляре команды `window.open()` для определения, будет или нет присутствовать панель каталога. Панель каталога содержит раздел BOOKMARKS. Используйте "yes", чтобы разрешить, "no" — запретить.
  - `height` является командой характеристики, которая обозначает высоту в пикселях нового окна или изображения.
  - `location` является характеристикой, используемой в экземпляре команды `window.open()` для указания, должна ли выводиться панель адреса в открываемом окне.

Используйте "yes", чтобы разрешить, "no" — запретить. Существует также команда `location`, которая действует как объект.

- `menubar` является характеристикой, используемой в экземпляре команды `window.open()` для определения, должна ли выводиться панель меню в открываемом окне. Панель меню является частью браузера, которая имеет кнопки FILE и EDIT. Используйте "yes", чтобы разрешить, "no" — запретить.
- `resizable` является характеристикой, используемой в экземпляре команды `window.open()` для определения, сможет ли пользователь изменять размер окна. Используйте "yes", чтобы разрешить, "no" — запретить.
- `scrollbars` является характеристикой, используемой в экземпляре команды `window.open()` для определения, будет ли присутствовать в новом окне строка прокрутки. Используйте "yes", чтобы разрешить, "no" — запретить.
- `status` является характеристикой, используемой в экземпляре команды `window.open()` для определения, будет ли присутствовать в новом окне строка состояния. Используйте "yes", чтобы разрешить, "no" — запретить. Существует также команда "status" которая действует как свойство.
- `toolbar` является характеристикой, используемой в экземпляре команды `window.open()` для указания, должна ли присутствовать панель инструментов в открываемом окне. Панель инструментов является частью браузера, которая имеет кнопки BACK (НАЗАД) и FORWARD (ВПЕРЕД). Используйте "yes", чтобы разрешить, "no" — запретить.
- `width` является командой характеристики, которая указывает ширину в пикселях нового окна или изображения.

- Функция аналогична присвоению имени переменной результату командной строки JavaScript. Фактически событию или последовательности событий присваивается имя, так что затем можно будет обратиться к этим событиям с помощью только одного этого имени функции, а не переписывать команды снова и

снова.

  - Операторы `if... else` являются условными операторами, которые действуют в зависимости от выбора пользователя из двух или большего количества исходов. `if` активируется, если ответом будет `true` (или "yes"), а `else` активируется, если ответом будет `false` (или "no").
  - Функция `eval()` используется для преобразования текста в число. Например, если пользователя просят ввести число, то сценарий все равно получает его в виде текста. Необходимо использовать эту функцию `eval()`, чтобы указать что текст на самом деле является числом.

- Экземпляр команды содержится в скобках, следующих сразу за командой. Экземпляр содержит данные о том, что должен сделать объект или как должен выполниться метод.

- Цикл. Существует фактически два вида циклов: `for` и `while`. `for` обычно используется, когда известно, сколько раз JavaScript должен выполнить свою функцию. `while` применяется, когда неизвестно, сколько раз JavaScript должен выполнить цикл.

  - `for` — это функция цикла, которая задает, сколько раз должен выполниться цикл.
  - `while` —- это команда цикла, которая используется, когда неизвестно, сколько раз JavaScript должен выполнить цикл, чтобы добиться требуемого результата.
  - Вложенный цикл является циклом `for` или `while`, заданным внутри другого цикла `for` или `while`. Фактически это цикл внутри цикла.

- Метод является командой, которая говорит, как надо поступить с объектом.

  - `alert` является методом, используемым в гипертекстовой ссылке или объекте окна для создания диалогового окна. Окно содержит текст, указанный в скобках `alert`, и кнопку OK, которую необходимо нажать, прежде чем пользователь сможет продолжить.

- `close` является методом, который действует на объект окна, чтобы закрыть текущее окно браузера.
- `confirm` является методом, который выводит диалоговое окно с кнопками OK и CANCEL. Команда используется наиболее часто с функциями `if` и `else`.
- `getMonth()` — метод объекта `Date`. Возвращает числовое представление месяца (`0` - `11`).
- `getDate()` — метод объекта `Date`. Возвращает числовое представление дня месяца (`1` — `31`).
- `getYear()` — метод объекта `Date`. Возвращает двузначное числовое представление года (`00` — `99`).
- `getDay()` — метод объекта `Date`. Возвращает числовое представление дня недели (`1` — `7`).
- `getHours()` — метод объекта `Date`. Возвращает числовое представление часа суток (`0` — `23`).
- `getMinutes()` — метод объекта `Date`. Возвращает числовое представление текущей минуты (`00` — `59`).
- `getSeconds()` — метод объекта `Date`. Возвращает числовое представление текущей секунды (`00` — `59`).
- `open` является методом, который открывает новый документ или новое окно.
- `prompt` является методом, который выводит пользователю диалоговое окно для ввода данных. Этот метод всегда сопровождается переменной, куда заносятся введенные пользователем данные.
- `write` является методом, который действует на объекте документа, для отправки текста на страницу.

- Объект (Object) является чем-то, что существует как документ HTML, окно браузера или дата и время. Объект также может быть чем-то, что создается с помощью функции.

  - `Date` является объектом, который необходимо назвать, чтобы иметь возможность вызвать методы даты и времени. Формат именования объекта следующий:

    ```
    Имя_Объекта = new Date();
    ```

Обратите внимание на слово `new` перед объектом `Date`.

- Document является объектом, который указывает на документ HTML, который содержит JavaScript.
- `history` является объектом, который указывает на память браузера страниц, посещенных пользователем. Список `history` может быть различного размера, в зависимости от того, сколько страниц посетил пользователь за время конкретного сеанса.

- `Location` является объектом, указывающим определенный URL. Он наиболее часто встречается в следующем формате:

  ```
  parent.location='index.html'
  ```

  Обратите внимание, что существует также свойство `location`, но оно действует совершенно другим образом.

- `Navigator` является объектом, определяющим браузер пользователя.
- `window` является объектом, указывающим экран браузера.

- Свойство (property) является характеристикой или частью большего объекта. Примером может быть строка состояния окна браузера, указывающая состояние окна (`object.property`)

  - `alinkColor` — свойство объекта `document`. Определяет цвет гипертекстовой ссылки, когда она активна или на ней был совершен щелчок мышью.
  - `appCodeName` — свойство объекта `navigator`. Определяет кодовое имя браузера, которое использует производитель.
  - `appName` — свойство объекта `navigator`. Указывает имя браузера, такое, как Netscape или Explorer
  - `appVersion` — свойство объекта `navigator`. Указывает номер версии и платформу браузера.
  - `bgColor` — свойство объекта `document`. Указывает цвет фона документа HTML.

- `defaultStatus` — свойство объекта `window`. Оно работает таким же образом, как `window.status`, за исключением того, что формат по умолчанию (default) используется, когда событие вызывается вне формата обработчика событий (встроенного в команду HTML формата).
- `fgColor` — свойство объекта `document`. Оно указывает цвет текста.
- `focus` — свойство любого объекта, которому вы хотите его присвоить. Свойство `focus` заставляет JavaScript рассматривать только один этот объект. Оно наиболее часто используется при проверке полей формы.
- `host` — свойство объекта `location`. Оно указывает URL сервера как текст или IP-адрес, в зависимости от того, как себе обозначает сам сервер. Кроме того свойство `port` возвращает номер порта. Эта команда очень похожа на свойство `hostname`, которое действует таким же образом, за исключением того, что не возвращает порт. Если порт `null` или не важен для сервера, то две команды выполняют одну функцию и будут возвращать одинаковые результаты.
- `hostname` — свойство объекта `location`. Оно указывает URL сервера как текст или IP-адрес, в зависимости от того, как себе обозначает сам сервер. Эта команда очень похожа на свойство `host`, которое действует таким же образом, за исключением того, что не возвращает порт. Если порт `null` или не важен для сервера, то две команды выполняют одну функцию и будут возвращать одинаковые результаты.
- `lastModified` — свойство объекта `document`. Оно указывает дату и время, когда страница была последний раз обновлена на сервере или последний раз сохранена на диск.
- `length` — свойство объекта `history`. Указывает число страниц, перечисленных в файле истории браузера. Перечисленные страницы являются страницами, которые посетил пользователь перед посещением страницы.
- `linkColor` — свойство объекта `document`. Оно указывает цвет гипертекстовых ссылок.
- `location` — свойство объекта `document`. Оно указывает URL страницы. Обратите внимание, что также имеется

объект `location`, который действует совершенно другим образом.

- `parent` — свойство, используемое обычно с фреймами, для указания определенной ячейки фрейма. Когда оно используется вне формата фрейма, оно указывает на все окно браузера.
- `referrer` — свойство объекта `document`. Оно указывает URL страницы, с которой только что пришел пользователь.
- `self` — свойство, которое указывает текущее окно.
- `status` — свойство объекта `window`. Оно обозначает строку состояния внизу экрана браузера.
- `target` является командой, которая располагается внутри гипертекстовой ссылки `HREF`. Ссылка указывает, куда должен направляться результат ссылки.
- `title` — свойство объекта `document`. Оно указывает текст между командами `<TITLE>` и `</TITLE>` в документе HTML.
- `top` — свойство объекта `window`. Оно указывает окно браузера самого верхнего уровня.
- `userAgent` — свойство объекта `navigator`. Оно указывает заголовок протокола передачи гипертекста, посланное для предупреждения сервера, какой браузер он обслуживает.
- `value` — считайте `value` значением того, что в определенное время что-то имеется или нет. Контрольный флажок может иметь значение on (включен) или off (выключен), в зависимости от того, был он отмечен или нет. Поле TEXT может иметь значение hidden (скрытое), если требуется, чтобы пользователь его не видел. И, как отмечалось выше, поле TEXT может содержать введенный текст. Он будет значением этого поля. К этому значению можно обратиться или передать в другую часть кода JavaScript для использования. Можно также вывести то, что пользователь ввел в текстовое поле.
- `vlinkColor` — свойство объекта `document`. Оно указывает цвет гипертекстовых ссылок после их посещения.

- Точка с запятой (;) является признаком конца оператора. Она

сообщает браузеру, что эта конкретная строка сценария завершилась. Если она не используется, то браузер будет считать, что строка продолжается, и возможна ошибка.

- Переменная (Variable) является именем, присвоенным элементу данных. Она позволяет определить односложное представление для целой строки кода. То есть, имя переменной представляет результат команды JavaScript. Это односложное имя можно использовать позже в сценарии в качестве представления результата целой строки кода, а не переписывать заново всю функцию JavaScript. Глобальная переменная является переменной, которая указана вне функции, так что любая следующая далее функция может ее использовать. Если переменная определена внутри функции, то только эта функция может ее использовать.

Содержание