

ВВЕДЕНИЕ В JAVASCRIPT ДЛЯ МАГА

© 1996, 1997 Стефан Кох (Stefan Koch)

Об этом руководстве

Online - версия

Данное руководство представляет собой введение в JavaScript. Я начал составлять его в качестве онлайн-руководства, где Вы бы имели возможность тут же проверять все описанные примеры на практике. Но поскольку подобное руководство стало со временем большим, понадобилось создать для него печатный вариант. Действительно, весьма утомительно читать большие куски текста, сидя перед монитором. Но вместе с тем очевидно, что печатное руководство не может в полной мере заменить онлайн-вариант. Поэтому у Вас есть также возможность найти онлайн-вариант по адресам <http://rummelplatz.uni-mannheim.de/~skoch/js/> или <http://www.webconn.com/java/javascript/intro> ("зеркало" в USA).

Книга по JavaScript и некоторые примеры

Недавно я написал книгу о JavaScript. Она называется 'JavaScript - Einfuehrung, Programmierung und Referenz' и написана на немецком языке. Для этой книги я создал специальную "домашнюю" страничку, которую можно найти по адресу <http://www.dpunkt.de/javascript/>

Здесь Вы найдете информацию о моей книжке и некоторые интересные примеры использования JavaScript. Материал на страницах представлен и на немецком, и на английском языках, так что смело изучайте примеры JavaScript, даже если Вы не знаете ни слова по-немецки.

Название: *JavaScript - Einfuehrung, Programmierung und Referenz* (german)

Автор: *Stefan Koch*

Издательство: *dpunkt.verlag*

ISBN: 3-920993-64-0

Homepage: <http://www.dpunkt.de/javascript/>

Часть 1: Первые шаги

Что такое JavaScript

JavaScript - новый язык для составления скриптов, разработанный фирмой Netscape. С помощью JavaScript Вы можете легко создавать интерактивные Web-страницы. В

данном руководстве Вы увидите, что можно сделать с помощью JavaScript, и даже более того - увидите, как это сделано.

JavaScript - это не Java!

Многие люди считают, что JavaScript - это то же самое, что и Java, лишь потому, что эти языки имеют схожие названия. На самом деле это **не** так. Я считаю, что сейчас будет излишне показывать Вам все различия между этими языками - так что запомните лишь то, что JavaScript - это *не* Java. Чтобы получить дополнительную информацию по затронутой теме, обратитесь пожалуйста к введению, опубликованному на сайте Netscape или в моей книге: -)

Запуск JavaScript

Что необходимо сделать, чтобы запускать скрипты, написанные на языке JavaScript? Вам понадобится браузер, способный работать с JavaScript - например Netscape Navigator (начиная с версии 2.0) или Microsoft Internet Explorer (MSIE - начиная с версии 3.0). С тех пор, как оба этих браузера стали широко распространены, множество людей получили возможность работать со скриптами, написанными на языке JavaScript. Несомненно, это важный аргумент в пользу выбора языка JavaScript, как средства улучшения ваших Web-страниц.

Конечно же, перед чтением данного руководства Вы должны познакомиться с основами другого языка - HTML. При этом, возможно, Вы обнаружите, что много хороших средств диалога можно создать, пользуясь лишь командами HTML. Чтобы получить дополнительную информацию о языке HTML, лучше всего инициировать поиск по ключевому слову 'html' на поисковом сервере Yahoo.

Размещение JavaScript на HTML-странице

Код скрипта JavaScript размещается непосредственно на HTML-странице. Чтобы увидеть, как делается, давайте рассмотрим следующий простой пример:

```
<html>
<body>
<br>
Это обычный HTML документ.
<br>
  <script language="JavaScript">
    document.write("А это JavaScript!")
  </script>
<br>
Вновь документ HTML.
</body>
</html>
```

С первого взгляда пример напоминает обычный файл HTML. Единственное новшество здесь - конструкция:

```
<script language="JavaScript">
```

```
document.write("А это JavaScript!")  
</script>
```

Это действительно код JavaScript. Чтобы видеть, как этот скрипт работает, запишите данный пример как обычный файл HTML и загрузите его в браузер, имеющий поддержку языка JavaScript. В результате Вы получите 3 строки текста:

```
Это обычный HTML документ.  
А это JavaScript!  
Вновь документ HTML.
```

Я должен признать, что данный скрипт не столь полезен - то же самое и более просто можно было бы написать на "чистом" языке HTML. Я всего лишь хотел продемонстрировать Вам тэг признака `<script>`. Все, что стоит между тэгами `<script>` и `</script>`, интерпретируется как код на языке JavaScript. Здесь Вы также видите пример использования инструкции `document.write()` - одной из наиболее важных команд, используемых при программировании на языке JavaScript. Команда `document.write()` используется, когда необходимо что-либо написать в текущем документе (в данном случае таким является наш HTML-документ). Так наша небольшая программа на JavaScript в HTML-документе пишет фразу "А это JavaScript!".

Браузеры без поддержки JavaScript

А как будет выглядеть наша страница, если браузер не воспринимает JavaScript? Браузеры, не имеющие поддержки JavaScript, "не знают" и тэга `<script>`. Они игнорируют его и печатают все стоящие вслед за ним коды как обычный текст. Иными словами, читатель увидит, как код JavaScript, приведенный в нашей программе, окажется вписан открытым текстом прямо посреди HTML-документа. Разумеется, это не входило в наши намерения. На этот случай имеется специальный способ скрыть исходный код скрипта от старых версий браузеров - мы будем использовать для этого тэг комментария из HTML - `<!-- -->`. В результате новый вариант нашего исходного кода будет выглядеть как:

```
<html>  
<body>  
<br>  
Это обычный HTML документ.  
<br>  
<script language="JavaScript">  
<!-- скрывает код от старых браузеров  
  
document.write("А это JavaScript!")  
  
// -->  
</script>  
<br>  
Вновь документ HTML.  
</body>  
</html>
```

В этом случае браузер без поддержки JavaScript будет печатать:

*Это обычный HTML документ.
Вновь документ HTML.*

А без HTML-тэга комментария браузер без поддержки JavaScript напечатал бы:

*Это обычный HTML документ.
document.write("А это JavaScript!")
Вновь документ HTML.*

Пожалуйста обратите внимание, что Вы не можете полностью скрыть исходный код JavaScript. То, что мы здесь делаем, имеет целью предотвратить распечатку кода скрипта на старых браузерах - однако тем не менее читатель сможет увидеть этот код посредством пункта меню *'View document source'*. Не существует также способа скрыть что-либо от просмотра в вашем исходном коде (и увидеть, как выполнен тот или иной трюк).

События

События и обработчики событий являются очень важной частью для программирования на языке JavaScript. События, главным образом, инициируются теми или иными действиями пользователя. Если он щелкает по некоторой кнопке, происходит событие "Click". Если указатель мыши пересекает какую-либо ссылку гипертекста - происходит событие MouseOver. Существует несколько различных типов событий. Мы можем заставить нашу JavaScript-программу реагировать на некоторые из них. И это может быть выполнено с помощью специальных программ обработки событий. Так, в результате щелчка по кнопке может создаваться выпадающее окно. Это означает, что создание окна должно быть реакцией на событие щелчка - Click. Программа - обработчик событий, которую мы должны использовать в данном случае, называется onClick. И она сообщает компьютеру, что нужно делать, если произойдет данное событие. Приведенный ниже код представляет простой пример программы обработки события onClick:

```
<form>  
<input type="button" value="Click me" onClick="alert('Yo')">  
</form>
```

(online-версия руководства позволит Вам проверить этот скрипт немедленно)

Данный пример имеет несколько новых особенностей - рассмотрим их по порядку. Вы можете здесь видеть, что мы создаем некую форму с кнопкой (как это делать - проблема языка HTML, так что рассматривать это здесь я не буду). Первая новая особенность - *onClick="alert('Yo')"* в тэге *<input>*. Как мы уже говорили, этот атрибут определяет, что происходит, когда нажимают на кнопку. Таким образом, если имеет место событие Click, компьютер должен выполнить вызов *alert('Yo')*. Это и есть пример кода на языке JavaScript (Обратите внимание, что в этом случае мы даже не пользуемся тэгом *<script>*). Функция *alert()* позволяет Вам создавать выпадающие окна. При ее вызове Вы должны в скобках задать некую строку. В нашем случае это

'Yo'. И это как раз будет тот текст, что появится в выпадающем окне. Таким образом, когда читатель когда щелкает на кнопке, наш скрипт создает окно, содержащее текст 'Yo'.

Некоторое замешательство может вызвать еще одна особенность данного примера: в команде `document.write()` мы использовали двойные кавычки (`"`), а в конструкции `alert()` - только одинарные. Почему? В большинстве случаев Вы можете использовать оба типа кавычек. Однако в последнем примере мы написали `onClick="alert('Yo')"` - то есть мы использовали и двойные, и одинарные кавычки. Если бы мы написали `onClick="alert("Yo")"`, то компьютер не смог бы разобраться в нашем скрипте, поскольку становится неясно, к которой из частей конструкции имеет отношение функция обработки событий `onClick`, а к которой - нет. Поэтому Вы и вынуждены в данном случае перемежать оба типа кавычек. Не имеет значения, в каком порядке Вы использовали кавычки - сперва двойные, а затем одинарные или наоборот. То есть Вы можете точно так же написать и `onClick='alert("Yo")'`.

Вы можете использовать в скрипте множество различных типов функций обработки событий. Сведения о некоторых из них мы получим в данном описании, однако не о всех. Поэтому обращайтесь пожалуйста к соответствующему справочнику, если Вы хотите узнать, какие обработчики событий еще существуют.

Итак, если Вы используете браузер Netscape Navigator, то выпадающее окно содержит текст, что был передан функции JavaScript `alert`. Такое ограничение накладывается по соображениям безопасности. Такое же выпадающее окно Вы можете создать и с помощью метода `prompt()`. Однако в этом случае окно будет воспроизводить текст, введенный читателем. А потому, скрипт, написанный злоумышленником, может принять вид системного сообщения и попросить читателя ввести некий пароль. А если текст помещается в выпадающее окно, то тем самым читателю дается понять, что данное окно было создано web-браузером, а не вашей операционной системой. И поскольку данное ограничение наложено по соображениям безопасности, Вы не можете взять и просто так удалить появившееся сообщение.

Функции

В большинстве наших программ на языке JavaScript мы будем пользоваться функциями. Поэтому уже теперь мне необходимо рассказать об этом важном элементе языка. В большинстве случаев функции представляют собой лишь способ связать вместе нескольких команд. Давайте, к примеру, напомним скрипт, печатающий некий текст три раза подряд. Для начала рассмотрим простой подход:

```
<html>
<script language="JavaScript">
<!-- hide
```

```
document.write("Добро пожаловать на мою страницу!<br>");
document.write("Это JavaScript!<br>");
```

```
document.write("Добро пожаловать на мою страницу!<br>");
document.write("Это JavaScript!<br>");
```

```
document.write("Добро пожаловать на мою страницу!<br>");
document.write("Это JavaScript!<br>");
```

```
// -->
</script>
</html>
```

И такой скрипт напишет следующий текст

Добро пожаловать на мою страницу!
Это JavaScript!

три раза. Если посмотреть на исходный код скрипта, то видно, что для получения необходимого результата определенная часть его кода была повторена три раза. Разве это эффективно? Нет, мы можем решить ту же задачу еще лучше. Как насчет такого скрипта для решения той же самой задачи?:

```
<html>
<script language="JavaScript">
<!-- hide
```

```
function myFunction() {
    document.write("Добро пожаловать на мою страницу!<br>");
    document.write("Это JavaScript!<br>");
}
```

```
myFunction();
myFunction();
myFunction();
```

```
// -->
</script>
</html>
```

В этом скрипте мы определили некую функцию, состоящую из следующих строк:

```
function myFunction() {
    document.write("Добро пожаловать на мою страницу!<br>");
    document.write("Это JavaScript!<br>");
}
```

Все команды скрипта, что находятся внутри фигурных скобок - {} - принадлежат функции myFunction (). Это означает, что обе команды document.write() теперь связаны воедино и могут быть выполнены при вызове указанной функции. И действительно, нашем примере есть три вызова этой функции - Можно увидеть, что мы написали строку myFunction() три раза сразу после того, как дали определение самой функции. То есть как раз и сделали три вызова. В свою очередь, это означает, что содержимое этой функции (команды, указанные в фигурных скобках) было выполнено трижды. Поскольку это довольно простой пример использования функции, то у Вас мог возникнуть вопрос, а почему собственно эти функции столь важны в JavaScript. По прочтении данного описания Вы конечно же поймете их пользу. Именно возможность

передачи переменных при вызове функции придает нашим скриптам подлинную гибкость - что это такое, мы увидим позже.

Функции могут также использоваться совместно с процедурами обработки событий. Рассмотрим следующий пример:

```
<html>
<head>

<script language="JavaScript">
<!-- hide

function calculation() {
    var x= 12;
    var y= 5;

    var result= x + y;

    alert(result);
}

// -->
</script>

</head>
<body>

<form>
<input type="button" value="Calculate" onClick="calculation()">
</form>

</body>
</html>
```

(online-версия руководства позволит Вам проверить этот скрипт немедленно)

Здесь при нажатии на кнопку осуществляется вызов функции *calculation()*. Как можно заметить, эта функция выполняет некие вычисления, пользуясь переменными *x*, *y* и *result*. Переменную мы можем определить с помощью ключевого слова *var*.

Переменные могут использоваться для хранения различных величин - чисел, строк текста и т.д. Так строка скрипта *var result= x + y;* сообщает браузеру о том, что необходимо создать переменную *result* и поместить туда результат выполнения арифметической операции *x + y* (т.е. *5 + 12*). После этого в переменный *result* будет размещено число 17. В данном случае команда *alert(result)* выполняет то же самое, что и *alert(17)*. Иными словами, мы получаем выпадающее окно, в котором написано число 17.