**JavaScript - Урок 1. Основные понятия**

Язык программирования JavaScript был разработан фирмой Netscape в сотрудничестве с Sun Microsystems и анансирован в 1995 году. JavaScript предназначен для создания интерактивных html-документов. Основные области использования JavaScript:

* Создание динамических страниц, т.е. страниц, содержимое которых может меняться после загрузки.
* Проверка правильности заполнения пользовательских форм.
* Решение "локальных" задач с помощью сценариев.
* JavaScript-код - основа большинства Ajax-приложений.

JavaScript позволяет создавать приложения, выполняемые на стороне клиента, т.е. эти приложения выполняются браузером на компьютере пользователя.  
  
Программы (сценарии) на этом языке обрабатываются встроенным в браузер интерпретатором. К сожалению, не все сценарии выполняются корректно во всех браузерах, поэтому тестируйте свои javascript-программы в различных браузерах.  
  
Язык JavaScript регистрозависимый, т.е. заглавные и прописные буквы алфавита считаются разными символами.  
  
Прежде, чем приступить к написанию сценариев, необходимо ознакомиться с основными понятиями, такими как литералы, переменные и выражения.  
  
***Литералы***  
  
Литералы - это простейшие данные с которыми может работать программа.

* *Литералы целого типа* - целые числа в представлении:
  + десятичном, например: 15, +5, -174.
  + шестнадцатеричном, например: 0х25, 0хff. Шестнадцатеричные числа включают цифры 0 - 9 и буквы a, b, c, d, e, f. Записываются они с символами 0х перед числом.
  + восьмеричном, например: 011, 0543. Восьмеричные числа включают только цифры 0 - 7.
* *Вещественные литералы* - дробные цисла. Целая часть отделяется от дробной точкой, например: 99.15, -32.45. Возможна экспоненциальная форма записи, например: 2.73e-7. В привычном виде это 2.73Х10-7, но в javascript знак умножения и 10 заменяются символом -e-.
* *Логические значения* - из два: истина (true) и ложь (false).
* *Строковые литералы* - последовательность символов, заключенная в одинарные или двойные кавычки. Например: "ваше имя", 'ваше имя'.

***Переменные***  
  
Переменные используются для хранения данных. Переменные определяются с помощью оператора *var*, после которого следует имя переменной. Имя переменной должно начинаться с буквы латинского алфавита или с символа подчеркивания. Само имя может включать буквы латинского алфавита, цифры и знак подчеркивания.  
  
Например:  
var test  
var \_test  
var \_my\_test1  
  
Каждой переменной можно присвоить значение либо при ее инициализации (объявлении), либо в коде самой программы. Оператор присваивания обозначается знаком равно (*=*), но знак равенства здесь не имеет своего прямого значения. В данном случае он лишь указывает на то, что данной переменной присвоили значение.  
  
Например:  
var a=15  
var b=23.15  
var c='выполнено'  
var s=true  
  
Каждая переменная имеет тип, определяемый значением переменной. Так в нашем примере: переменные *a* и*b*имеют тип *number*, переменная *c* имеет тип *string*, а переменная *s* - логический тип.

Команды определяющие тип данных

parseInt() – целое число

parseFloat() – вещественное число

***Выражения***  
  
Выражения строятся из литералов, переменных, знаков операций и скобок. В результате вычисления выражения получается единственное значение, которое может быть числом, строкой или логическим значением.  
  
В выражении a\*b, *a* и *b* называются *операндами*, а *\** - знак операции. В javascript определены следующие операции:

|  |  |
| --- | --- |
| **операция** | **название** |
| + | сложение |
| - | вычитание |
| \* | умножение |
| / | деление |
| % | остаток от деления целых чисел |
| ++ | увеличение значения операнда на единицу |
| -- | уменьшение значения операнда на единицу |

Операторы в выражении вычисляются слева направо в соответствии с приоритетами арифметических операций. Изменить порядок вычисления можно с помощью скобок.  
  
Для каждого из арифметических операторов есть форма, в которой одновременно с заданной операцией выполняется присваивание:

|  |  |
| --- | --- |
| **оператор** | **эквивалентный оператор присваивания** |
| X+=Y | X=X+Y |
| X-=Y | X=X-Y |
| X\*=Y | X=X\*Y |
| X/=Y | X=X/Y |
| X%=Y | X=X%Y |

В данном случае сначала вычисляется правый операнд, а затем полученное значение присваивается левому операнду.  
  
Для того, чтобы можно было сравнивать два значения в javascript определены операции сравнения, результатом которых может быть только логическое значение: true или false:

|  |  |
| --- | --- |
| **операция** | **название** |
| < | меньше |
| <= | меньше или равно |
| == | равно |
| != | не равно |
| >= | больше или равно |
| > | больше |

В javascript определены логические операции:  
  
*&&* - логическое И (AND),  
  
*||* - логическое ИЛИ (OR),  
  
*!* - логическое НЕ (NOT).  
  
Результаты воздействия логических операторов на различные комбинации значений операндов показаны в таблице:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **A&&B** | **A||B** | **!A** |
| true | true | true | true | false |
| true | false | false | true | false |
| false | true | false | true | true |
| false | false | false | false | true |

Проще говоря, значение выражения A&&B истинно, если истинны оба оператора, и ложно в противном случае. Значение выражения A||B истинно, если значение хотя бы одного операнда истинно, и ложно в противном случае. Если значение операнда А истинно, то !A - ложно и наоборот.  
  
Для строковых литералов определена операция *конкатенация строк*, т.е. их объединение. Обозначается эта операция знаком плюс (+). Результатом выполнения операции является также строка. Пример:  
  
var st1="Привет";  
var st2="Вася";  
var st3=st1+st2;   
  
В результате в переменной st3 будет значение "Привет Вася".  
  
В конце приведем таблицу приоритетов операций в порядке их убывания. Приоритет операций определяет порядок, в котором выполняются операции в выражении.

|  |  |
| --- | --- |
| **название** | **обозначение** |
| инкремент | ++ |
| декремент | -- |
| отрицание | ! |
| унарный минус | - |
| умножение | \* |
| деление, остаток от деления | /,% |
| сложение | + |
| вычитание | - |
| сравнение | <, >, <=, >= |
| равенство | == |
| не равенство | != |
| логическое И | && |
| логическое ИЛИ | || |
| присваивание | =, +=, -=, \*=, /=, %=, != |

# JavaScript - Урок 2. Где размещать javascript-код

Javascript-код может быть размещен как на самой странице, так и во внешнем файле.  
  
**Javascript-код на странице**  
  
Сценарии, написанные на языке javascript, могут располагаться на самой html-странице между тегами *<script></script>*, которые в свою очередь обычно находятся в тегах *<head></head>*. У тега *<script>* должен быть указан параметр *language*, который и указывает язык написания скрипта:

<html>

<head>

<title>Заголовок документа</title>

<link rel="stylesheet" type="text/css" href="style.css">

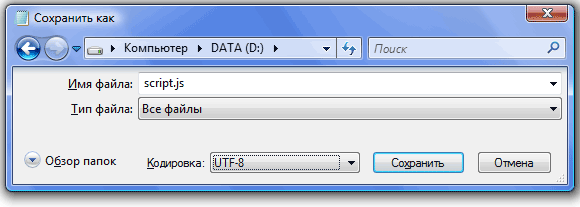
<script language="javascript"> </script>

</head>

<body>

</body>

</html>

Теоретически теги *<script></script>* могут располагаться и в теле документа (в тегах *<body></body>*). Выполняться такой сценарий будет тогда, когда браузер "дочитает" страницу до этого места (до тегов *<script></script>*). Использовать такой способ сейчас не принято.  
  
**Javascript-код во внешнем файле**  
  
Создайте в блокноте страницу (пока пустую) и сохраните ее как script.js в ту же папку, где лежит html-страница:  
  


Это будет наша страница с javascript-кодом. Теперь нам надо подключить страницу script.js к html-странице. Для этого, как вы помните, в html существует тег *<script>*, который и отвечает за подключение внешних файлов скриптов ([html - урок 2](http://www.site-do.ru/html/html2.php)). Добавим этот тег в html-страницу:

<html>

<head>

<title>Заголовок документа</title>

<link rel="stylesheet" type="text/css" href="style.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

</body>

</html>

# JavaScript – Урок 3. Обработчики событий

Давайте напишем нашу первую программу. Для этого сделаем исключение и поместим наш javascript-код на html-странице. Итак, откройте (или создайте) html-страницу и поместите в нее следующий код:

<html>

<head>

<title>Заголовок документа</title>

<link rel="stylesheet" type="text/css" href="style.css">

<script language="javascript">

document.write("Моя первая страница.");

</script>

</head>

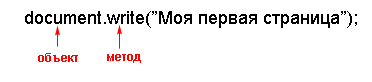
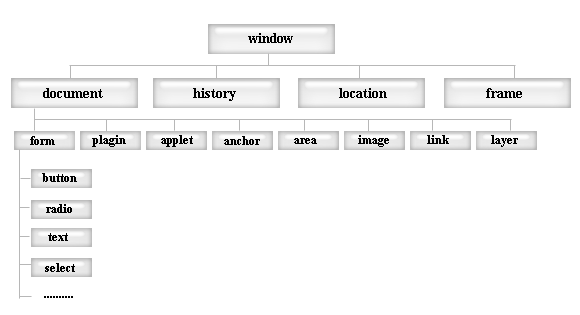
<body>

</body>

</html>

Откройте эту страницу в браузере, она выглядит так:

http://www.site-do.ru/images/js2.gif

Давайте разберемся как это работает. Браузер читает нашу html-страницу, видит оператор для выполнения *document.write("Моя первая страница.");* и выполняет его. Рассмотрим из чего состоит сам оператор (инструкция):  
  
  
  
При интерпретации html-страницы браузер создает объекты javascript. Они хранятся в виде иерархической структуры, отражая структуру документа, например:  
  


На самом верхнем уровне находится объект *window*, представляющий окно браузера и являющийся "родителем" всех остальных объектов. Расположенные ниже могут иметь свои подчиненные объекты. Так объект *document* (текущая страница) может иметь дочерний объект *form* (форма) и т.д.  
  
Все объекты имеют методы (отделяются от объекта точкой), например:  
*document.write* позволяет писать текст в текущую страницу,  
*window.open* открывает новое окно браузера.  
  
Также объекты имеют свойства, например:  
*document.bgcolor* содержит значение фонового цвета текущей страницы,  
*document.title* содержит заголовок страницы.  
  
Все инструкции программного кода заканчиваются точкой с запятой. Об этом надо помнить.  
  
Сценарий, который мы написали, выполнился немедленно после загрузки страницы. Однако в большинстве случаев нам нужно, чтобы сценарий выполнялся после каких-либо действий пользователя: нажатия на кнопку или ввода текста. Т.е. в ответ на пользовательское событие.  
  
Для этого в тегах элементов страницы введены параметры обработки событий, задающие действия, выполняемые при возникновении события, связанного с элементом. Например:

<div onClick="addText();"></div>

Здесь *Click* - событие (щелчок по div-у), *onClick* - обработчик события, *addText()* - имя функции, которая сработает при возникновении этого события (щелчка по div-у).  
  
Функции мы будем рассматривать в следующем уроке, а сейчас перечислим события, которые поддерживаются javascript. Запоминать их не следует, в дальнейшем можете просто обращаться к этой таблице.

|  |  |  |
| --- | --- | --- |
| **событие** | **когда происходит** | **обработчик события** |
| Blur | потеря объектом фокуса | onBlur |
| Change | пользователь изменяет значение элемента | onChange |
| Click | пользователь щелкает мыщью по объекту | onClick |
| DblClick | пользователь делает двойной щелчок мышью по объекту | onDblClick |
| DragDrop | пользователь перетаскивает мышью объект | onDragDrop |
| Error | возникновение javascript-ошибки | onError |
| Focus | окно или элемент формы получает фокус | onFocus |
| KeyDown | пользователь нажимает клавишу клавиатуры | onKeyDown |
| KeyPress | пользователь удерживает нажатой клавишу клавиатуры | onKeyPress |
| KeyUp | пользователь отпускает клавишу клавиатуры | onKeyUp |
| Load | документ загружается в браузер | onLoad |
| MouseDown | пользователь нажимает кнопку мыши | onMouseDown |
| MouseOut | указатель мыши выходит за пределы элемента | onMouseOut |
| MouseOver | указатель мыши помещается над элементом | onMouseOver |
| MouseUp | пользователь отпускает кнопку мыши | onMouseUp |
| Move | пользователь перемещает окно | onMove |
| Reset | пользователь нажимает кнопку "reset" формы | onReset |
| Resize | пользователь изменяет размеры окна или элемента | onResize |
| Select | пользователь выбирает элемент формы | onSelect |
| Submit | пользователь нажимает кнопку "submit" формы | onSubmit |
| Unload | пользователь закрывает документ | onUnload |

**JavaScript - Урок 4. Создание JavaScript-функций**

*Функция* - это именованная последовательность операторов (инструкций). Любая функция имеет следующий синтаксис:  
  
function имя (){  
оператор;  
.......  
оператор;  
}   
  
Сначала идет ключевое слово *function*, затем имя функции, затем в круглых скобках перечисляются параметры (если они есть), затем в фигурных скобках перечисляются операторы, т.е. последовательность выполняемых действий. Каждый оператор заканчивается точкой с запятой.  
  
Рассмотрим на примере. Во втором уроке мы создали html-страницу, страницу script.js для функций и подключили одну к другой. Эти страницы мы и будем использовать. Поместим в html-страницу следующий код:

<html>

<head>

<title>Заголовок документа</title>

<link rel="stylesheet" type="text/css" href="style.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

<div onClick="showMessage();">Щелкни меня</div>

</body>

</html>

Мы указали, что при щелчке пользователя по div-у (т.е. по словам "Щелкни меня") должна быть вызвана функция с именем "showMessage". Теперь на странице script.js надо написать саму функцию "showMessage". Открываем страницу script.js и пишем:

function showMessage(){

alert ("Вы щелкнули по div-у");

}

Наша функция "showMessage" будет выполнять одно действие - отображать окно предупреждений с текстом "Вы щелкнули по div-у". *alert* - это стандартная функция javascript, которая и выводит окно предупреждений, мы лишь задаем текст для этого окна.  
  
В javascript есть стандартный набор функций, который можно использовать (например, alert) и так называемые пользовательские функции, т.е. те, которые мы создаем сами. Некоторые функции javascript мы будем изучать по мере необходимости, некоторым же будут посвящены отдельные уроки.  
  
Вернемся к нашему примеру. Откройте вашу html-страницу с помощью браузера и щелкните по div-у. Можете щелкнуть и в примере ниже.

Щелкни меня

Таким образом, мы рассмотрели механизм работы обработки событий:

1. В html-странице мы привязываем к нужному элементу обработчик события (onClick, onMouseUp и т.д.), в качестве значения которого указываем имя функции, которая и должна сработать, когда настанет событие.
2. На странице с расширение .js мы пишем код этой функции, т.е. инструкции, которые должны быть выполнены, если функция инициализирована (к ней обратились).

Давайте усложним задачу и напишем сценарий, который будет рассчитывать площадь прямоугольника по введенным пользователем длине и ширине. Для этого сначала разместим на html-странице нужные элементы формы:

<html>

<head>

<title>Расчет площади прямоугольника</title>

<link rel="stylesheet" type="text/css" href="style.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

<form name="forma1">

Введите длину прямоугольника <input type="text" name="t1" size="10"><br><br>

Введите ширину прямоугольника <input type="text" name="t2" size="10"><br><br>

<input type="button" name="button" value="Вычислить"><br><br>

Площадь прямоугольника равна <input type="text" name="res" size="10">

</form>

</body>

</html>

В браузере наша страница будет выглядеть так:

Начало формы

Введите длину прямоугольника   
  
Введите ширину прямоугольника   
  
  
  
Площадь прямоугольника равна 

Конец формы

Если вы забыли, как задаются элементы форм, то обратитесь к уроку [HTML - работа с формами](http://www.site-do.ru/html/html11.php).  
  
Итак, пользователь вводит значения ширины и длины и нажимает на кнопку "Вычислить". После чего, в поле площадь должен появиться результат. Таким образом, событие наступает при нажатии на кнопку "Вычислить", значит именно к ней мы и привяжем обработчик события. Функцию вычисления площади назовем "areaRectangle":

.......

<input type="button" name="button" value="Вычислить"

onClick="areaRectangle();"><br><br>

.......

Теперь пришло время написать саму функцию "areaRectangle". Для этого откроем страницу script.js и напишем заготовку для функции:

function areaRectangle(){

}

Теперь надо написать тело функции. Для начала объявим три переменные: *a* - значение длины прямоугольника, *b* - значение ширины прямоугольника, *s* - площадь прямоугольника:

function areaRectangle(){

var a;

var b;

var s;

}

Значение (value) *a* должно браться из текущей страницы (document), из формы с именем "forma1", из текстового поля с именем "t1". Так это и записывается *document.forma1.t1.value*, т.е. перечисляются через точку имена объектов от родительского до нужного (иерархическую структуру объектов мы обсуждали в предыдущем уроке). Последним указывается необходимое свойство объекта (value).  
  
Аналогично и для значения *b* - *document.forma1.t2.value*.  
  
А наша переменная *s* - есть произведение *a* на *b*. Запишем это в тело функции:

function areaRectangle(){

var a=document.forma1.t1.value;

var b=document.forma1.t2.value;

var s=a\*b;

}

Осталось только написать инструкцию записи вычисленной площади в текстовое поле с именем "res" нашей формы. Т.е нам надо, чтобы в текщую страницу, в форму с именем "forma1", в текстовое поле с именем "res", в качестве значения (value) было присвоено значение *s*. Также добавим команду объявляющую переменные ***a*** и b, как целочисленные parseInt().

function areaRectangle(){

var a=parseInt(document.forma1.t1.value);

var b=parseInt(document.forma1.t2.value);

var s=a\*b;

document.forma1.res.value=s;

}

Иными словами, мы сначала присвоили нашим переменным *a* и *b* значения из формы, затем произвели необходимые расчеты, а после этого присвоили некоторому элементу формы полученное значение *s*.  
  
Проверьте работу нашей html-страницы у себя в браузере. Если вы все сделали правильно, то должно работать также, как в примере ниже:

Начало формы

Введите длину прямоугольника   
  
Введите ширину прямоугольника   
  
  
  
Площадь прямоугольника равна 

Конец формы

**JavaScript - Урок 5. Параметры функции**

На прошлом уроке мы написали сценарий, который вычисляет площадь прямоугольника по введенным пользователем значениям длины и ширины. Также мы написали функцию, которая собственно и производит вычисление. Но что, если у нас будет несколько html-страниц, на которых нужно будет вычислять площадь прямоугольника. Нам придется написать для каждой свою функцию? Если оставить сценарий в том виде, в котором он находится сейчас, то - ДА. Но это, как вы понимаете, не очень удобно.  
  
Разумнее написать один раз функцию и в дальнейшем использовать ее на всех html-страницах. Для этого html-страница должна каким-то образом указать функции, какие именно значения (с какой страницы) брать для вычисления. Вот здесь нам и пригодятся *параметры* (те, которые указываются в круглых скобках).  
  
Сейчас код нашей формы такой:

<html>

<head>

<title>javascript параметры</title>

<link rel="stylesheet" type="text/css" href="style.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

<form name="forma1">

Введите длину прямоугольника <input type="text" name="t1" size="10"><br><br>

Введите ширину прямоугольника <input type="text" name="t2" size="10"><br><br>

<input type="button" name="button" value="Вычислить" onClick="areaRectangle();"><br><br>

Площадь прямоугольника равна <input type="text" name="res" size="10">

</form>

</body>

</html>

А код нашей функции следующий:

function areaRectangle(){

var a=parseInt(document.forma1.t1.value);

var b=parseInt(document.forma1.t2.value);

var s=a\*b;

document.forma1.res.value=s;

}

Обратите внимание, в функции мы используем имя формы - forma1, его мы и сделаем параметром. Т.е. перепишем нашу функцию следующим образом:

function areaRectangle(obj){

var a= parseInt(obj.t1.value);

var b= parseInt(obj.t2.value);

var s=a\*b;

obj.res.value=s;

}

Т.е мы указали, что функция должна принять в качестве параметра какой-то объект (obj) и производить все действия с ним. Нам осталось на нашей html-странице указать имя объекта:

.......

<input type="button" name="button" value="Вычислить"

onClick="areaRectangle(forma1);"><br><br>

.......

Теперь, когда нам понадобится вычислить площадь на какой-либо другой странице, нам достаточно будет вызвать функцию areaRectangle(), а в скобках указать имя формы. Проверьте работу нашего сценария в браузере.  
  
В качестве параметров функции можно передавать любые значения, причем их может быть несколько. Рассмотрим, например, такую задачу. У нас есть три квадратика, при щелчке по каждому должно появляться окно сообщений с указанием цвета квадратика, по которому щелкнули (пощелкайте по квадратикам ниже).

|  |  |  |
| --- | --- | --- |
|  |  |  |

Итак, сначала на странице script.js напишем код функции, которая будет открывать окно сообщений, назовем ее "message". Как вы помните, вызов окна сообщений осуществляет стандартная функция javascript - *alert*, а текст сообщений мы будем передавать, как параметр, обозначим его как "m":

function message(m){

alert (m);

}

Теперь в html-странице напишем код самих квадратиков:

<html>

<head>

<title>javascript параметры</title>

<link rel="stylesheet" type="text/css" href="style.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

<table><tr>

<td><div id="red" onClick="message('Вы щелкнули по красному квадратику');"></div></td>

<td><div id="green" onClick="message('Вы щелкнули по зеленому квадратику');"></div></td>

<td><div id="blue" onClick="message('Вы щелкнули по синему квадратику');"></div></td>

</tr></table>

</body>

</html>

Мы указали, что при щелчке по нашим квадратикам должна срабатывать функция "message", а в качестве параметров указали текст нужных сообщений (они будут подставляться вместо параметра "m" при обращении к функции).  
  
Осталось только оформить наши квадратики. Для этого на странице style.css напишем стили для них:

#red, #green, #blue{

width:100px;

height:100px;

margin:10px;

}

#red{

background:red;

}

#green{

background:green;

}

#blue{

background:blue;

}

Все, осталось только проверить работу сценария в браузере.  
  
Рассмотрим еще один пример, объединяющий предыдущие. Пусть у нас есть список ягод, а при наведении мышкой на название ягоды, ее описание будет появляться в текстовом поле.

Начало формы



Конец формы

* Малина
* Черника
* Ежевика

Такая конструкция может пригодиться не на одной странице, поэтому нам потребуется два параметра: один с именем объекта (формы), второй - с описанием ягод. Параметры в таком случае записываются через запятую. Итак, наша функция будет выглядеть так:

function showDesc(obj, n){

obj.desc.value=n;

}

где desc - имя текстового поля для вывода описаний. Эта функция будет срабатывать, когда на название ягоды наведен курсор, но нам понадобится еще одна функция, которая будет очищать текстовое поле, когда курсор выйдет за пределы названия. Назовем ее "delete":

function delet(obj){

obj.desc.value=' ';

}

где пробел в одинарных кавычках означает пустую строку.  
  
Теперь напишем код самой страницы. Напоминаю, обработчик события, когда указатель мыши помещается над элементом, называется *onMouseOver*, а обработчика события, когда указатель мыши выходит за пределы элемента - *onMouseOut*.

<html>

<head>

<title>javascript параметры</title>

<link rel="stylesheet" type="text/css" href="style1.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

<form name="forma2">

<textarea name="desc" cols=45 rows=4></textarea>

</form>

<ul>

<li onMouseOver="showDesc(forma2,'Малина обыкновенная — кустарник с многолетним корневищем,'+

' из которого развиваются двухгодичные надземные стебли высотой до полутора метров.');"

onMouseOut="delet(forma2);">Малина</li>

<li onMouseOver="showDesc(forma2,'Черника — кустарничек высотой 15—30 см. Ветви отходят'+

' от главного стволика под острыми углами.');"

onMouseOut="delet(forma2);">Черника</li>

<li onMouseOver="showDesc(forma2,'Ежевика — название нескольких видов растений из рода Rubus'+

' семейства Розовые.');"

onMouseOut="delet(forma2);">Ежевика</li>

</ul>

</body>

</html>

Обратите внимание, текст в параметре заключается в одинарные кавычки, а при переносе строк используется операция конкатенации строк, т.е. каждая часть заключается в одинарные кавычки и эти части соединяются оператором *+*. У себя вы можете не использовать конкатенацию строк, просто напишите текст в одну строку.

# JavaScript – Урок 6. Объект math в javascript

Как уже было сказано, в javascript определены некоторые стандартные объекты и функции. Сегодня мы рассмотрим объект math и его методы.  
  
Объект math работает с математическими функциями, а его методы можно использовать для вызова этих функций. Ниже представлены некоторые методы объекта math:

|  |  |
| --- | --- |
| **метод** | **описание** |
| abs | абсолютное значение |
| sin, cos, tan | тригонометрические функции |
| log | натуральный логарифм |
| exp | экспонента |
| pow | показательная функция |
| sqrt | квадратный корень |
| min | наименьшее значение |
| max | наибольшее значение |

Какие-то методы вы будете использовать часто, какие-то редко, а некоторые вам и вовсе не пригодятся. Рассмотрим пример использования объекта math.  
  
Предположим мы хотим написать сценарий, который будет вычислять площадь треугольника по трем его сторонам. Для этого нам потребуется использовать формулу Герона:  
  
формула Герона  
  
Для тех, кто давно окончил школу, напоминаю:  
S - площадь  
a, b, c - длины сторон треугольника  
  
Итак, напишем в html-странице код формы:

<html>

<head>

<title>math javascript</title>

<link rel="stylesheet" type="text/css" href="style1.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

<form name="forma3">

Сторона 1 <input type="text" size="8" maxlength="8" name="st1">

Сторона 2 <input type="text" size="8" maxlength="8" name="st2">

Сторона 3 <input type="text" size="8" maxlength="8" name="st3"><br><br>

<input type="button" value="Вычислить" onClick="areaOfTriangle(forma3);">

<input type="reset" value="Отменить">

Результат <input type="text" size="8" maxlength="8" name="res">

</form>

</body>

</html>

Здесь нет ничего нового. Теперь на странице script.js напишем код функции:

function areaOfTriangle(obj){

var a=1\*obj.st1.value;

var b=1\*obj.st2.value;

var c=1\*obj.st3.value;

var p=(a+b+c)/2;

var s=Math.sqrt(p\*(p-a)\*(p-b)\*(p-c));

obj.res.value=s;

}

Здесь мы использовали объект *Math* и его метод *sqrt* для извлечения квадратного корня. Выражение, из которого извлекается корень, должно быть взято в скобки.  
  
Также обратите внимание на первые три строчки функции, они начинаются с *1\**, т.е. наши переменные a, b, c мы умножили на единицу. Зачем? Помните, в первом уроке мы говорили о том, что у каждой переменной есть тип, который определяется автоматически. Так как наши переменные приходят из текстового поля, то и тип они имеют *string*, т.е. они воспринимаются функцией не как числа, а как буквы.  
  
Если бы мы перемножали эти переменные, то их тип автоматически переопределился бы к *number*, но мы их складываем, а знак операции *+* расценивается в данном случае, как конкатенация строк. Чтобы этого не происходило, мы и умножили наши переменные на единицу, преобразовав их таким образом к типу *number*. Ради эксперимента уберите умножение на единицу в этих трех строках и посмотрите, что "насчитает" сценарий. Затем верните правильный вариант и убедитесь, что все работает правильно, как в примере ниже.

Начало формы

Сторона 1  Сторона 2  Сторона 3   
  
  Результат 

Конец формы

Вроде все хорошо, но есть один нюанс: если извлекаемый корень является дробным числом, то дробная часть может быть бесконечно длинной. Для точных вычислений это необходимо, но в большинстве случаев достаточно двух знаков после запятой.  
  
Чтобы округлить результат до N знаков после запятой можно воспользоваться методом *toFixed* объекта *Number*. Синтаксис записи следующий:

(x).toFixed(N)

где x - число, которое надо округлить, а N - число знаков после запятой.  
  
Давайте исправим конец нашей функции и округлим результат до 2 знаков после запятой:

function areaOfTriangle(obj){

var a=1\*obj.st1.value;

var b=1\*obj.st2.value;

var c=1\*obj.st3.value;

var p=(a+b+c)/2;

var s=Math.sqrt(p\*(p-a)\*(p-b)\*(p-c));

s=s.toFixed(2);

obj.res.value=s;

}

Вот теперь все аккуратненько:

Начало формы

Сторона 1  Сторона 2  Сторона 3   
  
  Результат 

Конец формы

# JavaScript - Урок 7. Ветвления в программе - оператор if

Очень часто возникает ситуация, когда нам нужно выполнить какое-либо действие в зависимости от какого-либо условия. Например, у нас интернет-магазин одежды. Мы спрашиваем у пользователя кто он (мужчина или женщина) и в зависимости от ответа отображаем список соответствующих товаров (мужских или женских). При написании подобных программ используется *условный оператор if*. Синтаксис у него следующий:  
  
if B {S1}  
else {S2}  
  
  
где *B* - выражение логического типа, а *S1* и *S2* - операторы.  
  
Работает это так: вычисляется значение выражения *B*, если оно истинно, то выполняется оператор *S1*, если оно ложно, то выполняется оператор *S2*. Строку *else {S2}*можно опустить.  
  
Думаю на примере будет понятнее. Пусть у нас есть форма, в которую пользователь вводит 3 значения. Напишем сценарий, который определит максимальное из введеных чисел.  
  
Для начала напишем в html-странице код формы:

<html>

<head>

<title>javascript if</title>

<link rel="stylesheet" type="text/css" href="style1.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

<form name="forma4">

Значение 1 <input type="text" size="8" maxlength="8" name="zn1">

Значение 2 <input type="text" size="8" maxlength="8" name="zn2">

Значение 3 <input type="text" size="8" maxlength="8" name="zn3"><br><br>

<input type="button" value="Максимальное значение" onClick="maxZnach(forma4);">

<input type="text" size="8" maxlength="8" name="res">

<input type="reset" value="Отменить">

</form>

</body>

</html>

Теперь на странице script.js напишем код функции:

function maxZnach(obj){

var a=1\*obj.zn1.value;

var b=1\*obj.zn2.value;

var c=1\*obj.zn3.value;

var m=a;

if (b>m) m=b;

if (c>m) m=c;

obj.res.value=m;

}

Итак, наша функция принимает три значения из формы, за максимальное (*m*) мы принимаем значение *a*. Затем мы сравниваем: если значение *b* больше максимального (т.е. *a*), то максимальным становится *b*, в противном случае максимальным остается *a* (т.к. выражение в скобках не является истинным). Далее аналогично сравниваем следующее значение *c* с максимальным. Ответ выводим в поле результата (*res*).

Начало формы

Значение 1  Значение 2  Значение 3   
  
  

Конец формы

Вообще такой сценарий можно было бы написать, используя метод *max* объекта *Math*, рассмотренного в прошлом уроке, и код получился бы короче:

function maxZnach(obj){

var a=1\*obj.zn1.value;

var b=1\*obj.zn2.value;

var c=1\*obj.zn3.value;

obj.res.value=Math.max(Math.max(a,b),c);

}

Это я к тому, что программирование все-таки процесс творческий, и одну задачу можно решить разными способами. Задача же программиста найти наиболее оптимальный вариант. Но это так, лирическое отступление. Вернемся к условному оператору if и рассмотрим более интересный пример. Напишем сценарий, во-время работы которого, при наведении курсора мыши на изображение, оно будет увеличиваться, создавая эффект приближения.  
  
Как вы помните в HTML можно задать размер вставляемого изображения. Если заданные размеры больше или меньше оригинала, то браузер автоматически подгонит оригинал под эти размеры. Этим мы и воспользуемся. Пусть у нас есть вот такая картинка:  
  
  
  
Ширина оригинала 302 пиксела. Мы же хотим, чтобы на странице картинка имела ширину 102 пиксела, а при наведении курсора, увеличивалась до 302 пикселов. С html-страницей все понятно:

<html>

<head>

<title>javascript if</title>

<link rel="stylesheet" type="text/css" href="style1.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

<img src="images/tigrenok.jpg" name="tigr" width="102" onMouseOver="bigPict()">

</body>

</html>

А в нашей функции, кроме условного оператора, мы воспользуемся еще стандартной функцией javascript *setTimeout*, которая вызывает пользовательскую функцию с заданным промежутком времени:

function bigPict(){

var w=document.tigr.width;

if (w<302){

document.tigr.width=w+10;

document.tigr.src="images/tigrenok.jpg"

setTimeout("bigPict()", 500)

}

}

Таким образом, функция проверяет ширину картинки (*width*) и, если она меньше 302 пикселов, то увеличивает эту ширину на 10 пикселов. Функция *setTimeout* вызывает нашу функцию *bigPict* каждые полсекунды, благодаря чему размер картинки будет увеличиваться до тех пор, пока условие *w<302* не станет ложным.



Чтобы визуально сделать приближение более плавным попробуйте уменьшить шаг увеличения ширины и время обращения к функции. Поиграйтесь с этими числами и найдите наиболее оптимальные.

# JavaScript - Урок 8. Оператор выбора switch

В предыдущем уроке мы познакомились с оператором *if*. Он удобен, когда нужно совершить какое-либо действие в зависимости от выбора пользователя. Но что если таких выборов будет много? Например, мы знаем прогноз погоды на неделю вперед. А выдаем прогноз только на один день, на тот который выберет пользователь. Как это сделать?  
  
Вариант первый - использовать оператор *if*.  
  
Для этого в html-странице напишем саму форму:

<html>

<head>

<title>javascript switch</title>

<link rel="stylesheet" type="text/css" href="style1.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

Прогноз погоды на какой-день Вас интересует:<br><br>

<form name="forma5">

<select name="day" size="7">

<option selected value="mon">понедельник

<option value="tue">вторник

<option value="wen">среда

<option value="thu">четверг

<option value="fri">пятница

<option value="sat">суббота

<option value="san">воскресенье

</select>

<input type="button" value="OK" onClick="Vday(forma5);">

<br><br>

<textarea cols="35" rows="5" wrap="virtual" name="res"></textarea>

</form>

</body>

</html>

Теперь на странице script.js напишем код функции:

function Vday(obj){

if (obj.elements[0][0].selected)

obj.res.value="В понедельник будет ветрено, температура воздуха поднимется до +20 градусов";

if (obj.elements[0][1].selected)

obj.res.value="Во вторник будет солнечно, температура воздуха поднимется до +25 градусов";

if (obj.elements[0][2].selected)

obj.res.value="В среду будет прохладно, температура воздуха опустится до +17 градусов";

if (obj.elements[0][3].selected)

obj.res.value="В четверг будет пасмурно, температура воздуха опустится до +10 градусов";

if (obj.elements[0][4].selected)

obj.res.value="В пятницу будет облачно, температура воздуха поднимется до +15 градусов";

if (obj.elements[0][5].selected)

obj.res.value="В субботу будет ветрено, температура воздуха поднимется до +27 градусов";

if (obj.elements[0][6].selected)

obj.res.value="В воскресенье будет ясно, температура воздуха поднимется до +30 градусов";

}

Здесь надо дать некоторые пояснения. А именно, как наш сценарий узнает, какой элемент выбран.  
  
В нашей форме есть три элемента: список, определяемый тегом *select*, кнопка *OK*, и многострочное текстовое поле.  
  
Сценарий нумерует все элементы формы сверху вниз, начиная с 0. Поэтому, способ обращения к нашему списку выглядит так:

obj.elements[0]

В списке все элементы также пронумерованы от 0 до 6 соответственно. А элемент, который выбран, обладает свойством *selected*. Поэтому строка

obj.elements[0][0].selected

означает, что выбран первый пункт первого элемента объекта (в нашем случае первый пункт списка из формы). Вообще это не самый удобный способ обращения к элементу, но об этом мы поговорим подробно в следующем уроке, а сейчас вернемся к примеру.

Прогноз погоды на какой-день Вас интересует:

Начало формы


Конец формы

Все работает. Предположим мы решили немного изменить условия задачи. Пусть теперь пользователь не выбирает день недели, а вводит его в текстовое поле.  
  
Код html-страницы будет выглядеть так:

<html>

<head>

<title>javascript switch</title>

<link rel="stylesheet" type="text/css" href="style1.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

Прогноз погоды на какой-день Вас интересует:<br><br>

<form name="forma6">

<input type="text" size="15" maxlength="15" name="zn">

<input type="button" value="OK" onClick="Vday1(forma6);"><br><br>

<textarea cols="35" rows="5" wrap="virtual" name="res"></textarea>

</form>

</body>

</html>

А в функции снова придется 7 раз прописать оператор *if*, да еще предусмотреть вариант, если пользователь допустит орфографическую ошибку:

function Vday1(obj){

var a=obj.zn.value;

if (a=="понедельник")

{obj.res.value="В понедельник будет ветрено, температура воздуха поднимется до +20 градусов";}

else

if (a=="вторник")

{obj.res.value="Во вторник будет солнечно, температура воздуха поднимется до +25 градусов";}

else

if (a=="среда")

{obj.res.value="В среду будет прохладно, температура воздуха опустится до +17 градусов";}

else

if (a=="четверг")

{obj.res.value="В четверг будет пасмурно, температура воздуха опустится до +10 градусов";}

else

if (a=="пятница")

{obj.res.value="В пятницу будет облачно, температура воздуха поднимется до +15 градусов";}

else

if (a=="суббота")

{obj.res.value="В субботу будет ветрено, температура воздуха поднимется до +27 градусов";}

else

if (a=="воскресенье")

{obj.res.value="В воскресенье будет ясно, температура воздуха поднимется до +30 градусов";}

else

obj.res.value="Укажите правильно день недели";

}

Прогноз погоды на какой-день Вас интересует:

Начало формы


Конец формы

В принципе все работает, но согласитесь, что код нашей функции слишком длинный. Чтобы его сократить, используется оператор выбора *switch*. Синтаксис у него следующий:  
  
switch (B)  
{  
case L1: S1;  
case L2: S2;  
... ... ...  
case Ln: Sn;  
default:S  
}   
  
где *B* - выражение, *L1, L2, ..., Ln* - литералы, а *S1, S2, ..., Sn* - операторы.  
  
Работает это так: вычисляется значение выражения *B*. Если *B* равно *L1*, то выполняется оператор *S1*. Если *B* равно *L2*, то выполняется оператор *S2* и т.д. Если *B* не равно ни одному из значений, то выполняется оператор *S*.  
  
Давайте перепишем код нашей последней функции с помощью оператора *switch*.

function Vday1(obj){

var a=obj.zn.value;

var s;

switch (a)

{

case 'понедельник': s="В понедельник будет ветрено, температура воздуха поднимется до +20 градусов";

break;

case 'вторник': s="Во вторник будет солнечно, температура воздуха поднимется до +25 градусов"; break;

case 'среда': s="В среду будет прохладно, температура воздуха опустится до +17 градусов"; break;

case 'четверг': s="В четверг будет пасмурно, температура воздуха опустится до +10 градусов"; break;

case 'пятница': s="В пятницу будет облачно, температура воздуха поднимется до +15 градусов"; break;

case 'суббота': s="В субботу будет ветрено, температура воздуха поднимется до +27 градусов"; break;

case 'воскресенье': s="В воскресенье будет ясно, температура воздуха поднимется до +30 градусов";

break;

default: s='Укажите правильно день недели'

}

obj.res.value=s;

}

Результат тот же, а код гораздо короче, да и читабельнее.  
  
Оператор *break* используется, чтобы обеспечить завершение работы переключателя после выполнения нужного варианта. Если его опустить ничего не изменится, просто с ним сценарий работает быстрее, хотя в данном примере это и незаметно.

# JavaScript – Урок 9. Объекты JavaScript

В [уроке 3](http://www.site-do.ru/js/js3.php) мы уже затрагивали эту тему. Сейчас поговорим об этом более подробно. В языке JavaScript все элементы на web-странице выстраиваются в иерархическую структуру. Каждый элемент представляется в виде объекта. И каждый такой объект может иметь определенные свойства и методы.  
  
Предположим, у нас есть вот такая web-страница:

http://www.site-do.ru/images/js6.gif

Перед регистрацией ознакомьтесь [с правилами](http://www.site-do.ru/js/js9.php).

Начало формы

|  |  |
| --- | --- |
| Форма регистрации | |
| Имя |  |
| e-mail |  |
| Пароль |  |
| Повтор пароля |  |
| Пол | мужской  женский |
| Увлечения |  |
| Ваши пожелания |  |
|  | |

Конец формы



Ее html-код:

<html>

<head>

<title>javascript if</title>

<link rel="stylesheet" type="text/css" href="style1.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

<img src="images/pict1.gif"><br><br>

<center>

Перед регистрацией ознакомьтесь <a href="#">с правилами</a>.<br><br>

<form name="forma1">

<table border="0" cellspacing="5" cellpadding="5">

<caption>Форма регистрации</caption>

<tr><td align="right" valign="top">Имя</td>

<td><input type="text" name="name" size="25"></td>

</tr>

<tr><td align="right" valign="top">e-mail</td>

<td><input type="text" name="e-mail" size="25"></td>

</tr>

<tr><td align="right" valign="top" >Пароль</td>

<td><input type="password" name="password" size="25"></td>

</tr>

<tr><td align="right" valign="top" >Повтор пароля</td>

<td><input type="password" name="password1" size="25"></td>

</tr>

<tr><td align="right" valign="top" >Пол</td>

<td><input type="radio" name="sex" value="man" checked>мужской

<input type="radio" name="sex" value="woman"> женский

</td>

</tr>

<tr><td align="right" valign="top">Увлечения</td>

<td><select name="hobby" size="7" multiple>

<option selected value="1">компьютеры

<option value="2">спорт

<option value="3">игры

<option value="4">животные

<option value="5">автомобили

<option value="6">клубы

<option value="7">музыка

</select>

</td>

</tr>

<tr><td align="right" valign="top">Ваши пожелания</td>

<td><textarea cols="30" rows="3" wrap="physical"></textarea>

</td>

</tr>

<tr><td align="right" colspan="2">

<input type="submit" name="submit" value="Отправить">

<input type="reset" name="reset" value="Очистить">

</td>

</tr>

</table>

</form>

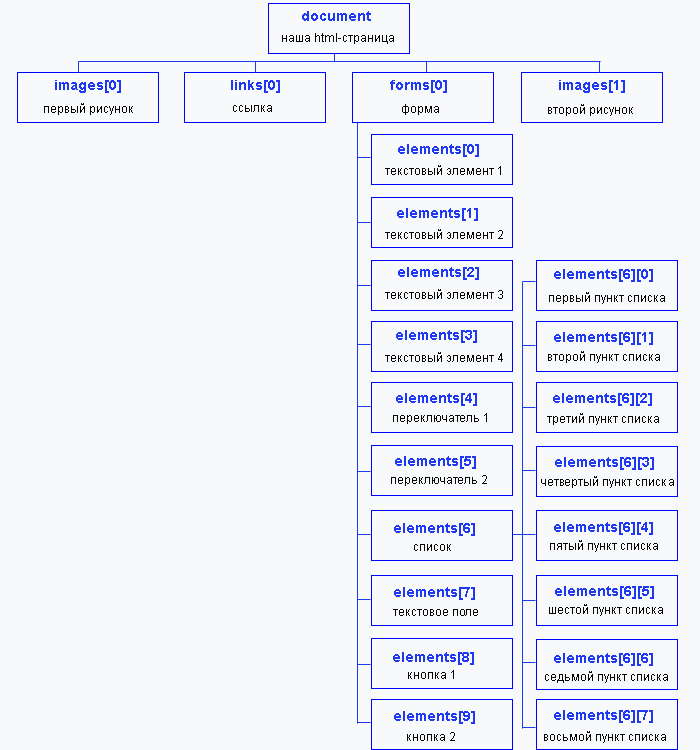
<img src="images/pict2.gif">

</center>

</body>

</html>

Иерархия объектов, создаваемая этой html-страницей, выглядит так:



Каждый объект этой структуры имеет свое имя и индекс. Индекс определяется положением объекта на странице (сверху вниз). Поэтому картинка, которая расположена выше на странице имеет меньший номер. Нумерация начинается с 0. Объект *forms*имеет свойство-массив - *elements*, в котором содержатся ссылки на элементы формы в порядке их перечисления в теге *form*. Их нумерация также начинается с 0. У объектов формы могут быть свои подчиненные объекты. В нашем примере у объекта *elements[6]*, который является списком, это пункты списка, которые имеют свою внутреннюю нумерацию.

Чтобы получить доступ к объекту, следует указать весь путь от вершины иерархии до нужного элемента. Так, путь к первой картинке выглядит так:

document.images[0]

Путь к форме:

document.forms[0]

Путь к списку:

document.forms[0].elements[6]

А путь к третьему пункту списка:

document.forms[0].elements[6][2]

Согласитесь, что такой подход не всегда удобен, так как легко запутаться во всех этих индексах. Поэтому всем элементам на странице обычно даются имена и доступ к элементам осуществляется по именам.  
  
Например, наша форма имеет имя forma1, а первое текстовое поле - имя name. Тогда, чтобы получить к нему доступ, мы должны написать:

document.forma1.name

Чтобы узнать, что пользователь ввел в это текстовое поле, используется свойство *value*:

document.forma1.name.value

Предположим, мы хотим осуществлять проверку на совпадение двух полей ввода пароля в нашем примере. Для этого напишем следующую функцию:

function prov(){

var a=document.forma1.elements[2].value;

var b=document.forma1.elements[3].value;

if (a==b){

alert ("Вы зарегистрированы!");

}

else

{

alert ("Введите правильный пароль.");

}

}

Пусть эта функция срабатывает при нажатии на кнопку "Отправить":

... ... ...

<input type="submit" name="submit" value="Отправить" onclick="prov();">

... ... ...

Чтобы узнать, какой пол указал пользователь, используется свойство переключателей *checked*. А для определения выбранного пользователем пункта из списка - свойство *selected*.  
  
Предположим, мы хотим, чтобы в зависимости от выбранных пола и увлечения, пользователю выдавалось какое-либо сообщение. Перепишем функцию prov():

function prov(){

if (document.forma1.elements[4].checked && document.forma1.elements[6][3].selected){

alert ("Мужчина, который любит животных заслуживает уважения.");

}

else if (document.forma1.elements[5].checked && document.forma1.elements[6][4].selected)

{

alert ("Женщина за рулем всегда вызывает интерес.");

}

else

{

alert ("Отличное увлечение.");

}

}

\* && - логическое И, было рассмотрено в [уроке 1](http://www.site-do.ru/js/js1.php) \*  
  
В JavaScript много объектов. Чтобы выяснить, какое из свойств и методов объекта позволяет получить доступ к нужной информации, необходимо обратиться к какому-либо справочнику по JavaScript, найти в нем нужный объект и посмотреть его свойства и методы. Целью же этого урока было показать на примере, как обратиться к объекту страницы и использовать информацию введенную пользователем.

# JavaScript - Урок 10. Циклы for и while

Циклы позволяют выполнять один и тот же код много раз и используются они очень часто.

## Цикл for

Самый популярный вид циклов - *цикл for*. Синтаксис у него следующий:  
  
for (A; B; I){S}   
  
где  
*A* - первоначальное выражение,   
*B* - условие продолжения. Если это выражение ложно, то выполнение цикла завершается,  
*I* - выражение инкремента,   
*{}* - тело цикла,   
*S* - операторы.  
  
Как всегда, на примере будет понятнее. Предположим, мы хотим написать сценарий, который будет складывать все числа от a до b, (где a и b будут вводиться пользователем) и отображать полученный результат.  
  
Итак, на html-странице мы будем иметь следующий код:

<html>

<head>

<title>javascript for</title>

<link rel="stylesheet" type="text/css" href="style.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

<form name="forma7">

Вас интересует сумма всех чисел

от <input type="text" name="a1" size="5" maxlength="5">

до <input type="text" name="a2" size="5" maxlength="5">

<input type="button" value="Вычислить" onclick="summa(forma7);"><br>

Сумма равна <input type="text" name="result" size="10" maxlength="10">

</form>

</body>

</html>

Выглядеть она будет так:

Начало формы

Вас интересует сумма всех чисел от  до    
Сумма равна 

Конец формы

Осталось на странице script.js написать саму функцию *summa()*:

function summa(obj) {

var summa=0;

var a1=1\*obj.a1.value;

var a2=1\*obj.a2.value;

for (var i = a1; i <= a2; i++) {

summa+=i;

}

obj.result.value = summa;

}

Из нового здесь только сам цикл, разберемся, как он работает:  
  
параметру цикла *i* присваивается значение *a1*, т.е. число, которое указал пользователь в текстовом поле *ОТ* и тело цикла (в фигурных скобках) выполняется с этим значением, т.е. в переменную *summa* записывается значение равное *a1*.   
  
После этого вычисляется выражение *i++*, т.е. значение *i* увеличивается на 1.  
  
Затем javascript проверяет условие *i <= a2* и, если оно истинно, тело цикла выполняется снова, т.е. значение переменной *summa* увеличивается на само себя плюс 1.  
  
Таким образом, цикл выполняется до тех пор, пока не сложит все числа от *a1* до *a2*.  
  
Когда условие *i <= a2* станет ложным, цикл прекратит свою работу, а полученный результат (значение переменной*summa*) будет выведен в поле *result*.  
  
Обратите внимание, что переменную *i* можно объявить прямо в цикле - *var i = a1*  
  
В общем, на этом можно было бы и закончить, но все-таки есть еще один нюанс. Для правильной работы нашего цикла значение *a2* должно быть больше значения *a1*. Добавим это условие в нашу функцию:

function summa(obj) {

var summa=0;

var a1=1\*obj.a1.value;

var a2=1\*obj.a2.value;

if (a2 > a1){

for (var i = a1; i <= a2; i++) {

summa+=i;

}

obj.result.value = summa;

}

else

alert("Значение ОТ должно быть меньше значения ДО")

}

Вот теперь наш цикл будет работать как надо.

## Цикл while

Не менее популярный вид циклов - *цикл while*. Синтаксис у него следующий:  
  
while (B){S}   
  
где  
*B* - условие продолжения. Если это выражение ложно, то выполнение цикла завершается,  
*{}* - тело цикла,   
*S* - операторы.  
  
Иными словами, операторы *S*, выполняются до тех пор, пока условие *B* не станет ложным.

# JavaScript - Урок 11. Дата, представление и обработка

В JavaScript дата определяется количеством миллисекунд, прошедших с 1 января 1970 года.  
  
Для работы с датой и временем применяется встроенный объект *Date*. Этот объект не имеет свойств, но обладает несколькими методами, позволяющими устанавливать и изменять дату и время.  
  
Объект *Date* создается с помощью оператороа *new* и конструктора - *Date*.  
  
Например:

var myData=new Date();

значением переменной *myData* будет текущая дата и время:

Sun Feb 12 2017 18:26:43 GMT+0300 (RTZ 2 (зима))

Методами объекта *Date* можно получать отдельно значения месяца, дня недели, часов, минут и секунд:

* *getDate* - возвращает число в диапазоне от 1 до 31, представляющее число месяца.
* *getHours* - возвращает час суток в диапазоне от 0 (полночь) до 23.
* *getMinutes* - возвращает минуты в диапазоне от 0 до 59.
* *getSeconds* - возвращает секунды в диапазоне от 0 до 59.

Предположим, мы хотим написать сценарий, который будет определять текущее время и выводить его в формате "чч:мм:сс".  
  
Код html-страницы будет простой:

<html>

<head>

<title>javascript дата</title>

<link rel="stylesheet" type="text/css" href="style.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

<form name="forma8">

<input type="button" value="Время" onClick="nTime(forma8);">

<input type="text" size="10" name="res">

</form>

</body>

</html>

Теперь напишем саму функцию *nTime()*:

function nTime(obj) {

var t=new Date();

var h=t.getHours();

var m=t.getMinutes();

var s=t.getSeconds();

var result=h+":"+m+":"+s;

obj.res.value=result;

}

\* как вы помните методы отделяются от объекта точкой, об этом мы говорили в [уроке 3](http://www.site-do.ru/js/js3.php) \*  
  
Как видите, все просто. Сначала определяем текущее время, а потом с помощью методов извлекаем из него отдельные значения часов, минут и секунд.

Начало формы



Конец формы

Здесь еще хочется пояснить строку *var result=h+":"+m+":"+s*. Впервые мы столкнулись с необходимостью выводить в результат, как значения переменных, так и простой текст. В принципе ничего сложного: переменные пишутся как есть, текст берется в кавычки, а знак *+* осуществляет операцию конкатенации, т.е. их объединения.  
  
В нашем примере остался один недочет, мы хотели, чтобы время выводилось в формате "чч:мм:сс", а сейчас оно выводится в формате "ч:м:с". Т.е., в 5 утра, время будет отображаться как "5:0:0", а хотелось бы так: "05:00:00" (что более привычно). В качестве домашнего задания можете попробовать это исправить. Например, с помощью оператора *if* и строкового литерала "0" (идея проста: если часов меньше 10, то в результат перед *h* написать "0" и так со всеми переменными).  
  
А пока продолжим изучать методы объекта *Date*:

* *getDay* - возвращает день недели, как целое число от 0 (воскресенье) до 6 (суббота).
* *getMonth* - возвращает номер месяца в году, как целое число от 0 (январь) до 11 (декабрь).
* *getYear* - возвращает год в виде двух последних цифр (*getFullYear* - возвращает год в виде четырех цифр).  
    
  \* К сожалению, начиная с 2000 года, есть проблема с отображением года в разных браузерах. Метод getYear в IE отображает полный год (вместо двух последних цифр), а FireFox вместо XX отображает 1XX (т.е. подставляет 1). Поэтому предпочтительнее использовать метод getFullYear.

Давайте напишем сценарий, который будет определять текущую дату и выводить ее в формате "число месяц год".  
  
Код html-страницы будет простой:

<html>

<head>

<title>javascript дата</title>

<link rel="stylesheet" type="text/css" href="style.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

<form name="forma9">

<input type="button" value="Дата" onClick="tData(forma9);">

<input type="text" size="20" name="res">

</form>

</body>

</html>

Теперь напишем саму функцию *tData()*:

function tData(obj) {

var s;

var t=new Date();

var y=t.getFullYear();

var d=t.getDate();

var mon=t.getMonth();

switch (mon)

{

case 0: s="января"; break;

case 1: s="февраля"; break;

case 2: s="марта"; break;

case 3: s="апреля"; break;

case 4: s="мае"; break;

case 5: s="июня"; break;

case 6: s="июля"; break;

case 7: s="августа"; break;

case 8: s="сентября"; break;

case 9: s="октября"; break;

case 10: s="ноября"; break;

case 11: s="декабря"; break;

}

var result=d+" "+s+" "+y;

obj.res.value=result;

}

Получилось подлиннее, чем в первом примере, т.к. приходится переводить на русский язык названия месяцев.

Начало формы



Конец формы

Рассмотренные выше методы позволяют получать дату. Если же нам потребуется установить дату, то следует использовать следующие методы:

* *setDate* - устанавливает число месяца в диапазоне от 1 до 31.
* *setHours* - устанавливает час для текущего времени в диапазоне от 0 (полночь) до 23.
* *setMinutes* - устанавливает минуты в диапазоне от 0 до 59.
* *setSeconds* - устанавливает секунды в диапазоне от 0 до 59.
* *setYear* - устанавливает значение года.
* *setMonth* - устанавливает значение месяца в диапазоне от 0 (январь) до 11 (декабрь).
* *setTime* - устанавливает значение объекта *Date* и возвращает количество миллисекунд, прошедших с 1 января 1970 года.

Так, если нам потребуется задать дату 06 декабря 2010 года, в функции мы будем иметь следующий код:

...

var t=new Date();

var y=t.setYear(2010);

var d=t.setDate(6);

var mon=t.setMonth(11);

...

Задать дату можно прямо в конструкторе, указав в качестве параметра строку формата "месяц, день, год часы:минуты:секунды":

var t=new Date("Feb,10,1975 17:45:10");

Значение часов, минут и секунд можно опустить (они будут равны нулю):

var t=new Date("Feb,10,1975");

Эту же дату можно задать с помощью чисел, перечислив через запятую год, месяц, число, часы, минуты, секунды:

var t=new Date(75, 1, 10, 17, 45, 10);

Или же, опуская часы, минуты и секунды (они будут равны нулю):

var t=new Date(75, 1, 10);

\* Есть проблема: IE год отображать не хочет, поэтому лучше эти варианты не использовать.

# JavaScript - Урок 12. Массивы

Массив представляет собой набор элементов, доступ к которым осуществляется по индексу. Создается массив с помощью оператора *new* и конструктора массива - функции *Array*.  
  
Предположим, что требуется создать массив имен пользователей. Это можно сделать так:

var users = new Array("Artem", "Irina", "Sergey", "Boris");

Это выражение создает массив из 4 элементов (пользователей). Все элементы массива, пронумерованы, начиная с нуля. Для получения значения элемента массива необходимо задать имя массива и в квадратных скобках порядковый номер элемента (его индекс). Так, для получения доступа к первому элементу нашего массива, можно написать так:

users[0]

Значения массива не обязательно задавать сразу. Можно, например, воспользоваться такой конструкцией:

var users = new Array(4);

Это выражение так же создает массив из 4 элементов, но значения элементов можно указать позже с помощью оператора присваивания:

users[0] = "Artem";

users[1] = "Irina";

users[2] = "Sergey";

users[3] = "Boris";

И, наконец, можно использовать конструктор без параметра:

var users = new Array();

В этом случае мы указываем, что создаваемая переменная *users* будет массивом, размер которого будет определяться автоматически по мере появления элементов.  
  
Чтобы определить длину массива (т.е. сколько элементов в нем находится) используется свойство *length*. Например, получить доступ к последнему элементу массива (если мы не знаем, сколько в нем элементов) можно следующим образом:

users[users.length-1];

Т.е. сначала определяется длина массива (*users.length*), затем, помня о том, что нумерация элементов начинается с нуля, из длины вычитается 1 и полученное значение используется в качестве индекса массива.  
  
Рассмотрим пример: предположим, мы хотим, чтобы при загрузке страницы пользователь видел какой сегодня день недели.  
  
Код html-страницы будет следующим:

<html>

<head>

<title>javascript массивы</title>

<link rel="stylesheet" type="text/css" href="style.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body onload="showDay();">

</body>

</html>

\* в [уроке 3](http://www.site-do.ru/js/js3.php) мы рассматривали события и их обработчики. В данном случае мы будем использовать событие Load (документ загружается в браузер) и его обработчик - onload. Иными словами, наша функция showDay() должна сработать при загрузке документа.\*  
  
Теперь напишем саму функцию *showDay()*:

function showDay() {

var nDays=new Array("воскресенье", "понедельник", "вторник", "среда", "четверг", "пятница",

"суббота");

var now=new Date();

var day=now.getDay();

var iDay=nDays[day];

var str="Сегодня - " + iDay;

document.write(str);

}

Рассмотрим подробнее. Сначала, мы создаем массив *nDays* из семи элементов (в предыдущем уроке, мы говорили о том, что нулю соответствует воскресенье):

var nDays=new Array("воскресенье", "понедельник", "вторник", "среда", "четверг", "пятница",

"суббота");

Затем, из текущей даты получаем текущий день недели (его числовой эквивалент):

var now=new Date();

var day=now.getDay();

и используем его в качестве индекса для обращения к элементу массива:

var iDay=nDays[day];

Последние две строки формируют и отображают результат:

var str="Сегодня - " + iDay;

document.write(str);

Попробуйте работу сценария в действии и убедитесь, что при загрузке страницы, появляется фраза типа "Сегодня - пятница".  
  
Двигаемся дальше. Два массива можно объединить в один с помощью метода *concat()*. Пусть у нас имеются два массива *a* и *b* и мы хотим их объединить в один - *c*:

var a=new Array(1, 2, 3);

var b=new Array(4, 5, 6);

var c=a.concat(b);

На выходе получим массив *c* из шести элементов: 1, 2, 3, 4, 5, 6.  
  
Массивы можно сортировать с помощью метода *sort()*, правда только в лексикографическом порядке (т.е. как строки). Например, если применить метод *sort()* к массиву с именами:

var users = new Array("Artem", "Irina", "Sergey", "Boris");

var c=users.sort();

то на выходе получим массив *c*: Artem, Boris, Irina, Sergey.  
  
А если применить метод *sort()* к массиву с числами:

var n = new Array(11, 12, 10, 107, 3, 20, 25, 101, 14, 34, 44, 5, 4);

var c=n.sort();

то на выходе получим следующий массив: 10, 101, 107, 11, 12, 14, 20, 25, 3, 34, 4, 44, 5. Т.е. элементы сравниваются как строки, а не как числа. Так что применять метод *sort()* надо осторожно.  
  
Теперь разберем на примере, как осуществить поиск заданного элемента в массиве. Предположим, у нас есть массив из 6 имен и мы хотим узнать, есть ли в нем определенное имя.  
  
Код html-страницы будет простой:

<html>

<head>

<title>javascript массивы</title>

<link rel="stylesheet" type="text/css" href="style.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

<form name="forma10">

Введите имя для проверки:

<input type="text" size="20" name="name">

<input type="button" value="Проверить" onClick="proverka(forma10);">

</form>

</body>

</html>

Теперь напишем саму функцию *proverka()* на странице script.js:

function proverka(obj) {

var k;

var users = new Array("Artem", "Irina", "Sergey", "Boris", "Olga", "Viktor");

var n=obj.name.value;

for (var i=0; i<=users.length-1; i++){

if (users[i]==n)

{k=0; break}

}

if (k==0){alert("Это имя присутствует в массиве.")}

else {alert("Такого имени в массиве нет.")}

}

Здесь мы воспользовались алгоритмом, который называется *линейный поиск*, согласно которому все элементы массива *users* сравниваются поочередно с образцом (с *n*, т.е. с именем, которое ввел пользователь). Если очередной элемент совпадает с образцом, то задача решена (цикл прерывается). Если же искомого имени в массиве нет, то мы узнаем об этом, лишь просмотрев все элементы массива.

Начало формы

Введите имя для проверки: 

Конец формы

В качестве домашнего задания напишите сценарий, который будет определять максимальный элемент числового массива.

**JavaScript работа со строками**

Мы уже не раз использовали строковые литералы, которые представляют собой последовательность символов, заключенную в одинарные или двойные кавычки. Все строки являются объектами типа *String* и с ними можно производить некоторые действия, например, мы уже знакомы с операцией конкатенации, т.е. объединения строк. В этом уроке посмотрим другие возможности работы со строками.   
Для начала определимся с некоторыми понятиями:

* *Алфавит* - конечное множество символов.
* *Строка* - конечная последовательность символов некоторого алфавита.
* *Пустая строка* - строка, не содержащая ни одного символа.

Чтобы создать строковый объект можно воспользоваться конструктором *newString*. Например:

var s = newString("Итого:");

Если применить единственное свойство объекта *String* - *length*, то мы узнаем длину строки.  
Для нашего примера, результатом выполнения *s.length* будет число 6.  
  
Рассмотрим методы, влияющие на строку:

* *charAt(n)* - возвращает символ, позицию которого определяет параметр *n*.  
  Для нашего примера, результатом выполнения *s.charAt(0)* будет буква "И".
* *substr(n1,n2)* - возвращает подстроку из строки, где *n1* - позиция первого символа подстроки, а *n2* - количество символов в строке.  
  Для нашего примера, результатом выполнения *s.substr(0, 4)* будет подстрока "Итог".

Для примера напишем сценарий, который будет определять, сколько раз заданное слово встречается в определенном тексте.  
  
Код html-страницы будет следующим:

<html>

<head>

<title>javascript строки</title>

<link rel="stylesheet" type="text/css" href="style.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

<form name="forma11">

Введите текст:<br>

<textarea name="textin" rows="4" cols="20"></textarea><hr>

Введите слово: <input type="text" size="10" name="slovo"><hr>

<input type="button" value="Определить" onClick="numword(forma11);">

<input type="reset" value="Отменить"><hr>

Количество слов в тексте: <input type="text" size="10" name="res">

</form>

</body>

</html>

Теперь напишем саму функцию *numword()* на странице script.js:

function numword(obj) {

var t=obj.textin.value;

var s=obj.slovo.value;

var m=s.length;

var res=0;

var i=0;

while (i < t.length-1)

{var ch=t.substr(i,m)

if (ch==s){

res+=1;

i=i+m

}

else

i++

}

obj.res.value=res

}

Посмотрим на результат, а затем разберем написанное:

Начало формы

Введите текст:  


Введите слово: 



Количество слов в тексте: 

Конец формы

Итак, посмотрим что же мы написали в нашей функции.  
Сначала мы определили две строковые переменные, одной является текст, введенный пользователем, другой - слово.

var t=obj.textin.value;

var s=obj.slovo.value;

Затем мы определили длину искомого слова и обнулили переменную с результатом:

var m=s.length;

var res=0;

Далее с помощью цикла while, мы указали следующее: пока параметр цикла *i* меньше либо равен длине текста (один вычитаем, т.к. нумерация символов начинается с нуля), брать подстроку, начиная с текущего символа *i* и длинной равной длине искомого слова и сравнивать ее с самим искомым словом. Если результат истинен, то переменная *res*увеличивается на единицу, а цикл продолжает свою работу с символа, который следует за найденным словом (*i=i+m-1*). В противном случае - со следующего символа:

var i=0;

while (i<=t.length-1)

{var ch=t.substr(i,m)

if (ch==s){

res+=1;

i=i+m-1

}

else

i++

}

Наконец, мы отображаем результат:

obj.res.value=res

В языке javascript определен ряд стандартных функций работы со строками:

* *Number(s)* - преобразует строковый параметр *s* в число.
* *String(n)* - преобразует число *n* в строку.
* *isNaN(s)* - проверяет является ли параметр *s* числом. Если параметр *s* не является числом, то возвращает значение *true* (истина), в противном случае - *false* (ложь).

Помните, когда в предыдущих уроках, мы брали числа, введенные пользователем, и умножали на 1. Таким образом, мы их из строки переводили в число. Например:

var a1=1\*obj.a1.value;

То же самое мы могли бы записать, используя стандартную функцию *Number(s)*:

var a=obj.a1.value;

var a1=Number(a);

# JavaScript - Урок 14. Регулярные выражения

Тема регулярных выражений достаточно объемная и в одном уроке ее не охватить, но цель наших уроков дать Вам основные представления о языке javascript и его возможностях, поэтому и обойти стороной регулярные выражения никак нельзя.  
  
Для начала разберемся, что же это такое.   
**Регулярное выражение** - это инструкция, описывающая на специально разработанном языке (RegExp) закон "сходства" искомой строки с образцом.  
  
Для чего это нужно? Например:

* Для организации поиска в тексте чего-либо.
* Для замены одной части подстрок другими.
* Для проверки корректности пользовательского ввода (наверно, вы не раз сталкивались с ситуацией, когда вводили адрес своей электронной почты в какую-либо форму и получали ошибку типа "Некорректный e-mail").

Не будем вдаваться в подробности, а посмотрим как задавать регулярные выражения. Существует два способа, в данной уроке мы рассмотрим один (создание в литеральной нотации):  
  
var p=/pattern/flags;   
  
где  
*pattern* - шаблон, является основой регулярного выражения, определяющей критерии сопоставления строк. Состоит из литералов и метасимволов.  
*flags* - флаги (модификаторы), задают дополнительные параметры сопоставления шаблона.  
  
Пример:  
  
var par=/[0-9a-z]+/i;   
  
Здесь *[0-9a-z]+* - шаблон, буквально означающий следующее "любое количество цифр и букв 1 и более раз" (как задавать шаблон посмотрим чуть ниже).  
  
*i* - флаг, означающий, что регистр символов не имеет значение.  
  
Чтобы было понятнее о чем речь, рассмотрим пример. Предположим, что у нас есть форма, куда пользователь вводит свой e-mail и пароль. Мы хотим, чтобы при нажатии на кнопку "Зарегистрировать" осуществлялась проверка корректности ввода.  
  
Код html-страницы будет следующим:

<html>

<head>

<title>javascript регулярные выражения</title>

<link rel="stylesheet" type="text/css" href="style.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

<h2>Форма регистрации</h2>

<form name="forma12">

<label>E-mail:</label><br>

<input type="text" name="mail"><br>

<label>Password:</label><br>

<input type="password" name="pas"><br>

<input type="button" value="Зарегистрировать" onclick="prov\_adress(forma12);">

</form>

</body>

</html>

Итак, что же должна сделать функция *prov\_adress()*? Для начала нам нужны две переменные, в которые мы поместим значения, введенные пользователем:

function prov\_adress(obj) {

var adr=obj.mail.value;

var par=obj.pas.value;

}

Теперь нам нужно задать образцы (регулярные выражения), с которыми мы и будем сравнивать то, что ввел пользователь. Здесь, я их просто приведу, о том, как их составлять поговорим позже:

function prov\_adress(obj) {

var adr=obj.mail.value;

var par=obj.pas.value;

var adr\_pattern=/[0-9a-z\_]+@[0-9a-z\_]+\.[a-z]{2,5}/i;

var par\_pattern=/[0-9a-z]+/i;

}

Теперь осуществляем проверку на сопоставление образцу. Для этого воспользуемся методом *test* объекта *RegExp*:

function prov\_adress(obj) {

var adr=obj.mail.value;

var par=obj.pas.value;

var adr\_pattern=/[0-9a-z\_]+@[0-9a-z\_]+\.[a-z]{2,5}/i;

var par\_pattern=/[0-9a-z]+/i;

var prov=adr\_pattern.test(adr);

var prov1=par\_pattern.test(par);

}

Строка *adr\_pattern.test(adr)* означает следующее: проверить существование в строке *adr* последовательности, сопоставляющейся с регулярным выражением *adr\_pattern*. Метод *test* возвращает логическое значение (true или false).  
  
Нам осталось только указать в нашей функции, что делать в случае успешной (или неуспешной) проверки:

function prov\_adress(obj) {

var adr=obj.mail.value;

var par=obj.pas.value;

var adr\_pattern=/[0-9a-z\_]+@[0-9a-z\_]+\.[a-z]{2,5}/i;

var par\_pattern=/[0-9a-z]+/i;

var prov=adr\_pattern.test(adr);

var prov1=par\_pattern.test(par);

if (prov==true && prov1==true) {

alert("Вы зарегистрированы!");

}

else {

alert("Введенные данные некорректны!");

}

}

Готово, надеюсь, суть того, что мы делаем вы поняли. Но прежде, чем проверить работу нашего сценария, посмотрим, из чего же состоят наши регулярные выражения.  
  
Возьмем регулярное выражение для нашего пароля - */[0-9a-z]+/i*:

* */[0-9a-z]+/* - шаблон, в котором:
  + *0-9* - любая цифра.
  + *a-z* - любая строчная буква от a до z.
  + *[]* - квадратные скобки означают, что в шаблоне может присутствовать любой из перечисленных в них литералов (в нашем случае цифры и строчные буквы)
  + *+* - указывает, что данная часть шаблона (т.е. то, что в квадратных скобках) может повторяться один и более раз.
* *i* - флаг, означающий, что регистр символов не имеет значение.

Иными словами, наше регулярное выражение указывает, что пароль может содержать любое количество цифр и букв 1 и более раз (т.е. он может состоять из одной цифры, из одной буквы, из многих цифр, из многих букв, из цифр и букв).   
  
Например, если пользователь введет в поле пароля "2", "a3b" или "leopard", то такой пароль будет считаться корректным. А если, он введет "ab&s" или "24?", то такой пароль корректным считаться не будет, т.к. он содержит спецсимволы, а мы их в регулярном выражении не разрешили.  
  
Надеюсь теперь стало понятно как и зачем можно использовать регулярные выражения, осталось узнать принципы их составления. Собственно говоря, задача составления регулярного выражения сводится к созданию его шаблона. А шаблон, как вы помните, может состоять из литералов и метасимволов.  
  
Начнем с самого простого - литералов:

* Каждый из таких символов представляет сам себя. Например, /abc/ - такому шаблону сопоставима только строка "abc".
* *a-z* - все строчные буквы от a до z. Например, /a-z/ - такому шаблону сопоставляются 26 строк: "a", "b", "c"... "z"
* *A-Z* - все заглавные буквы от A до Z.
* *0-9* - все цифры.

Если мы хотим указать, что цифр или букв может быть несколько, то придется использовать управляющие символы:

* *\** - указывает, что символ (или часть шаблона, если она заключена в квадратные скобки) может повторяться 0 и более раз. Например, /ab\*c/ - означает, что строка начинается с символа a, затем может быть сколько угодно символов b, после которых следует символ c. Т.е. это могут быть, например, такие строки: "ac", "abc", "abbbbbbc" и т.д.
* *+* - указывает, что символ (или часть шаблона, если она заключена в квадратные скобки) может повторяться 1 и более раз. Например, /ab+c/ - означает, что строка начинается с символа a, затем может быть сколько угодно символов b (но не меньше 1), после которых следует символ c. Т.е. это могут быть, например, такие строки: "abc", "abbbbbbc" и т.д.
* *.* - указывает, что на этом месте может быть любой одиночный символ, кроме символа новой строки. Например, для шаблона /ab.c/ сопоставимы такие строки: "ab6c", "abxc", "ab=c" и т.д.
* *?* - указывает, что символ (или часть шаблона, если она заключена в квадратные скобки) может повторяться 0 или 1 раз. Например, /ab?c/ - означает, что строка начинается с символа a, затем может быть или не быть один символ b, после которого следует символ c. Т.е. это могут быть такие строки: "ac", "abc"
* *{n}* - указывает, что символ (или часть шаблона, если она заключена в квадратные скобки) может повторяться ровно n раз. Например, /ab{3}c/ - означает, что строка начинается с символа a, затем идут 3 символа b, после которых следует символ c. Т.е. это будет строка "abbbc".
* *{n,}* - указывает, что символ (или часть шаблона, если она заключена в квадратные скобки) может повторяться n и более раз. Например, /ab{3,}c/ - означает, что строка начинается с символа a, затем идут 3 или более символов b, после которых следует символ c. Т.е. это могут быть такие строки: "abbbc", "abbbbbbbc" и т.д.
* *{n, m}* - указывает, что символ (или часть шаблона, если она заключена в квадратные скобки) может повторяться от n до m раз. Например, /ab{1,3}c/ - означает, что строка начинается с символа a, затем идут от 1 до 3 символов b, после которых следует символ c. Т.е. это могут быть такие строки: "abc", "abbc", "abbbc".
* *[]* - такой шаблон сопоставим с любым одиночным символом, принадлежащим множеству, определенному в скобках. Множество задается перечислением или указанием диапазона. Например, шаблону /[abc]/ могут быть сопоставимы строки: "a", "b", "c".
* *[^]* - такой шаблон сопоставим с любым одиночным символом, не принадлежащим множеству, определенному в скобках. Например, шаблону /[^abc]/ могут быть сопоставимы строки: "f", "x", "Z", но не могут быть сопоставимы строки: "a", "b", "c".
* *^* - указывает, что символы сопоставимы с началом строки. Например, шаблону /^abc/ могут быть сопоставимы строки: "abcd", "abcfh", но не могут быть сопоставимы строки: "dabc", "cbabc" и т.д.
* *$* - указывает, что символы сопоставимы с концом строки. Например, шаблону /abc$/ могут быть сопоставимы строки: "dabc", "fhabc", но не могут быть сопоставимы строки: "abcd", "abccb" и т.д.
* *|* - указывает на несколько альтернативных шаблонов. Например, шаблону /ab|c/ будут сопоставимы строки: "ab" и "c".
* *\* - служит для экранирования специальных символов, т.е. обратный слэш перед символом указывает на то, что он должен интерпретироваться как специальный. Например:  
    
  + *\d* - соответствует любая цифра от 0 до 9.
  + *\D* - соответствует все, кроме цифры.
  + *\s* - соответствует пробел.
  + *\S* - соответствует все, кроме пробела.
  + *\w* - соответствует буква, цифра или знак подчеркивания.
  + *\W* - соответствует все, кроме буквы, цифры или знака подчеркивания.

Например, шаблону /x\d\d/ будут соответствовать строки: "x01", "x25" и т.д., но не будут соответствовать строки: "A15", "x0A"...  
  
Также обратный слэш используется, чтобы специальный символ сделать литеральным. Например, если нам нужно найти строку "a\*b", то мы зададим следующий шаблон /a\\*b/.

Используя вышеперечисленные литералы и метасимволы, можно составлять какие угодно шаблоны (считай регулярные выражения). Давайте, например, посмотрим что же мы написали для шаблона e-mail в нашем примере:

var adr\_pattern=/[0-9a-z\_]+@[0-9a-z\_]+\.[a-z]{2,5}/i;

Итак, мы указали, что в адресе электронной почты идут цифры, буквы и знаки подчеркивания 1 и более раз, затем следует символ @, затем снова идут цифры, буквы и знаки подчеркивания 1 и более раз, затем идет символ точки, после которой идут буквы от 2 до 5 раз. Примерно такой вид имеют адреса электронной почты.  
  
Теперь, зная, что именно мы задали в образце, можно проверить работу примера:

## Форма регистрации

Начало формы

E-mail:  
  
Password:  


Конец формы

Попробуйте ввести e-mail вида "serega@mail" и посмотрите, что будет.  
  
Надо сказать, что составление шаблонов это своего рода искусство. К тому же, одно и то же регулярное выражение можно записать по-разному. Например, /[0-9a-z\_]/ - это тоже самое, что и /\w/.  
  
Если хотите потренироваться, то можете усовершенствовать регулярное выражение для проверки e-mail в нашем примере (оно далеко от совершенства).

# JavaScript - Урок 15. Работа с окнами

Работу с окнами мы рассмотрим на примере.  
  
Пусть у нас имеется страница, на которой расположены кнопки с названиями автомобилей и мы хотим, чтобы по щелчку по кнопке открывалось окно с картинкой этого автомобиля.   
  
Код html-страницы будет следующим:

<html>

<head>

<title>javascript окно</title>

<link rel="stylesheet" type="text/css" href="style.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

<form name="forma13">

<input type="button" value="ford" onClick="open\_ford()">

<input type="button" value="mazda" onClick="open\_mazda()">

<input type="button" value="volvo" onClick="open\_volvo()">

</form>

</body>

</html>

Теперь нам надо создать три страницы с изображениями автомобилей (ford.html, mazda.html, volvo.html), которые собственно и будут загружаться в новые окна.  
  
Код страницы ford.html будет следующим:

<html>

<head>

<title>javascript окно</title>

<link rel="stylesheet" type="text/css" href="style.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

<img src="images/ford.jpg" width="300">

<input type="button" value="Закрыть" onClick="close\_pict()">

</body>

</html>

Создайте аналогично две другие страницы самостоятельно (и не забудьте поместить в папку images соответствующие картинки).  
  
В наших функциях мы будем использовать два метода объекта window - *open()* и *close()*. Первый создает новое окно, второй - закрывает его.  
  
На странице script.js напишем код функций, а затем разберем его:

function open\_ford() {

ford=window.open("ford.html", "display\_ford",

"width=400,height=300,status=no,toolbar=no,menubar=no");

}

function open\_mazda() {

mazda=window.open("mazda.html", "display\_mazda"

"width=400,height=300,status=no,toolbar=no,menubar=no");

}

function open\_volvo() {

volvo=window.open("volvo.html", "display\_volvo"

"width=400,height=300,status=no,toolbar=no,menubar=no");

}

function close\_pict() {

window.close();

}

Давайте посмотрим, что мы написали. В первых трех функциях идут строки вида:

=window.open("ford.html", "display\_ford", "width=400,height=300,status=no,toolbar=no,menubar=no");

Эта строка создает новое окно с помощью метода *open()* объекта *window* (вы же помните, что методы отделяются точкой) и записывает в него страницу ford.html.  
  
У этого метода существует три параметра, каждый из которых заключается в кавычки:   
  
- первый параметр указывает страницу, которую нужно загрузить в окно (например, ford.html),  
  
- второй параметр задает имя открываемому окну (в нашем примере display\_ford),  
  
- третий параметр позволяет нам управлять процессом создания окна. Здесь можно задать размеры окна, указать, должно ли новое окно иметь строку статуса, панель инструментов или меню. В нашем примере, мы создали окно шириной 400 и высотой 300 пикселов, без строки статуса, панели инструментов и меню. А могли бы вместо *no*написать *yes* и тогда все бы это появилось в нашем окне.  
Здесь же можно указать ряд других параметров:  
*location* выводить строку URL   
*scrollbars* добавлять линейку прокрутки, если документ не помещается на экране   
*resize* позволять пользователю изменять размер окна   
Если какие-либо параметры опущены (в нашем примере, это три только что перечисленных), то из значение приравнивается к *no*. Обратите внимание, что все параметры перечисляются через запятую, НО БЕЗ ПРОБЕЛА!  
  
В последней функции, мы использовали метод *close()*, который закрывает текущее окно.  
  
Посмотрите на пример в действии:

Начало формы

Конец формы

У объекта *window*, кроме методов *open()* и *close()*, как вы понимаете, есть и другие. С одним из них, методом *alert ()*, мы уже сталкивались не раз. Давайте посмотрим, какие еще полезные методы есть у этого объекта:  
  
**confirm**  
  
Показывает диалоговое окно с указанным сообщением и кнопками "OK" и "Cancel". Сообщение должно побуждать пользователя принять решение. Подтверждение возвращает методу *true*, если пользователь выбирает "OK" или *false*если пользователь выбирает "Cancel".   
  
Синтаксис:  
  
confirm ("сообщение")   
  
Пример:  
  
пусть у нас есть какая-нибудь кнопка "Открыть окно" и мы хотим, чтобы по щелчку по ней появлялось диалоговое окно, позволяющее уточнить действительно ли мы хотим его открыть (для примера пусть открывается окно с изображением Volvo из предыдущего примера).

Начало формы

Конец формы

Как видите, теперь окно с картинкой открывается только в том случае, если мы выбрали "OK" в диалоговом окне.  
  
Код html-страницы для этого примера прост:

<html>

<head>

<title>javascript окно</title>

<link rel="stylesheet" type="text/css" href="style.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

<form name="forma14">

<input type="button" value="Открыть окно" onClick="choiceOf();">

</form>

</body>

</html>

На странице script.js напишем код функции choiceOf():

function choiceOf(){

if (confirm("Вы действительно хотите открыть окно?")) {

volvo=window.open("volvo.html", "display\_volvo",

"width=400,height=300,status=no,toolbar=no,menubar=no");

}

}

Т.е. в функции мы указали, что окно следует открывать только в случае, если пользователь в диалоговом окне выбрал "OK" (т.е. метод вернул истину (true)).  
  
Хотелось бы пояснить один нюанс, в программировании в условии оператора *if*, если метод возвращает либо истину (true), либо ложь (false), часть ==true опускается. Иными словами вместо:

(confirm("Вы действительно хотите открыть окно?")==true)

пишется:

(confirm("Вы действительно хотите открыть окно?"))

**prompt**  
  
Этот метод отображает диалоговое окно ввода пользователя.  
  
Синтаксис:  
  
prompt(сообщение, [значение по умолчанию])   
  
где  
*[]* - означают, что параметр необязателен, т.е. его можно опустить.   
  
Пример:  
  
Нажмите на кнопку ниже:

Начало формы

Конец формы

Я думаю, вы догадались, что мы хотим сделать. Мы хотим, чтобы пользователь ввел свое имя в диалоговом окне, после чего мы поздороваемся с ним по имени. \* По необъяснимым причинам этот пример в IE7 работает через раз. Если у Вас не сработало, то откройте этот урок, например, в FireFox.\*  
  
Итак, код html-страницы как всегда прост:

<html>

<head>

<title>javascript окно</title>

<link rel="stylesheet" type="text/css" href="style.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

<form name="forma15">

<input type="button" value="Давай знакомиться!" onClick="acquaintance()">

</form>

</body>

</html>

Впрочем, код функции тоже несложен:

function acquaintance(){

var YouName=prompt("Как тебя зовут?", "Напиши здесь свое имя.");

alert("Привет "+YouName);

}

Как видите, здесь мы использовали два метода объекта *window*: сначала метод *prompt*, который вернул введенное пользователем имя в переменную YouName, а затем - метод *alert*, который отобразил окно сообщений, в котором мы и поздоровались с пользователем по имени.  
  
**setTimeout**  
  
Этот метод делает что-либо по истечении указанного в миллисекундах промежутка времени.   
  
Синтаксис:  
  
setTimeout (что делать, время в миллисекундах)   
  
Пример:  
  
Пусть при нажатии на некоторую кнопку "Можно начинать?", через 3 секунды появляется окно сообщений с текстом "Начинайте!".

Начало формы

Конец формы

Итак, код html-страницы:

<html>

<head>

<title>javascript окно</title>

<link rel="stylesheet" type="text/css" href="style.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

<form name="forma16">

<input type="button" value="Можно начинать?" onClick="setTimeout('startMessage()',3000)">

</form>

</body>

</html>

Да, именно здесь мы и применим этот метод. Так мы укажем, что функция startMessage() должна сработать через 3 секунды.  
  
Код самой функции прост:

function startMessage(){

alert("Начинайте!")

}

Для того, чтобы отключить задержку таймера, установленную с помощью метода *setTimeout* используется следующий метод объекта *window*:   
  
**clearTimeout**  
  
Этот метод отключает таймер, установленный при помощи метода setTimeout.   
  
Синтаксис:  
  
clearTimeout (timerID)   
  
где  
*timerID* - уникальный идентификатор таймера, полученный при его установке.   
  
Давайте немножно переделаем наш последний пример, а именно сделаем две кнопки "Можно начинать?" и "Отменить вопрос":

Начало формы

Конец формы

Теперь, если нажать на кнопку "Можно начинать?", а затем, не дожидаясь появления окна сообщений, нажать на кнопку "Отменить вопрос", то окно сообщений не появится вовсе.  
  
При этом мы изменили только html-код страницы на следующий:

<html>

<head>

<title>javascript окно</title>

<link rel="stylesheet" type="text/css" href="style.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

<form name="forma17">

<input type="button" value="Можно начинать?" onClick="timer1=setTimeout('startMessage()',3000)">

<input type="button" value="Отменить вопрос" onClick="clearTimeout(timer1)">

</form>

</body>

</html>

Обратите внимание, что действие метода *setTimeout* происходит однократно. Если же мы хотим, чтобы какое-либо событие повторялось через определенные промежутки времени, то нам понадобится следующий метод:   
  
**setInterval**  
  
В отличие от *setTimeout*, этот метод выполняет код много раз, через равные промежутки времени, пока не будет остановлен при помощи метода *clearInterval*.  
  
Синтаксис:  
  
timerID=setInterval(что делать, время в миллисекундах)   
  
где  
*timerID* - уникальный идентификатор таймера, полученный при его установке.   
  
Давайте применим этот метод к предыдущему примеру:

<html>

<head>

<title>javascript окно</title>

<link rel="stylesheet" type="text/css" href="style.css">

<script type="text/javascript" src="script.js"></script>

</head>

<body>

<form name="forma18">

<input type="button" value="Можно начинать?" onClick="timer2=setInterval('startMessage()',3000)">

<input type="button" value="Я понял!" onClick="clearInterval(timer2)">

</form>

</body>

</html>

Теперь посмотрите, как это работает (прежде, чем нажимать на кнопку "Я понял", убедитесь, что окно сообщений открывается каждые 3 секунды):

Начало формы

Конец формы

Ну вот собственно и все, мы рассмотрели методы объекта *Window*, надеюсь они вам пригодятся в вашей работе.  
  
Собственно, на этом подошли к концу и наши уроки по JavaScript. Конечно, мы не рассмотрели все возможности этого языка, но как показывает практика, этого и не нужно. Все самое необходимое мы узнали, вы вполне в состоянии писать сценарии на этом языке.   
  
Практика, практика и еще раз практика, а если в процессе у вас возникнут вопросы, то ответы на них без труда можно найти при помощи того же поисковика Google. Целью же наших уроков было ввести Вас в мир JavaScript и показать его основные возможности.  
  
Но не спешите закрывать эту страницу. Как правило, сайтостроителей не интересуют детали реализации того или иного сценария, гораздо больше интересует возможность использовать уже готовые скрипты. Многие пользуются такими шаблонами, даже не вникая в основы JavaScript, что на мой взгляд не совсем правильно (ведь если что-то начнет работать не так, у таких любителей не будет возможности разобраться и поправить).  
  
Другое дело, если использовать специальные библиотеки, которые заметно облегчают жизнь. Что такое библиотека? Это некий набор многократно используемых функций.  
  
Ведь понятно, что для сайта, например, нужны графические эффекты. Вот кто-то постарался, написал, проверил и выложил это для всеобщего использования. Т.е. кто-то уже написал и протестировал в работе практически весь код, выполняющий определенные действия. Такие функции и объединяют в библиотеки. От вас же требуется только научиться использовать эти библиотеки для своих нужд. И тогда, может оказаться, что знаний, полученных в этих уроках по JavaScript вам более, чем достаточно.  
  
Наиболее популярными библиотеками JavaScript являются *Prototype* и *jQuery*. Обусловлено это тем, что они предоставляют кросс-браузерный интерфейс к методам DOM (объектная модель документа). Т.е. используя эти библиотеки, Вы вряд ли столкнетесь с ситуацией, когда в одном браузере сценарий работает, а в другом - нет. А это очень большой плюс.