# JavaScript

JavaScript представляет собой технологию, которая позволяет сделать страницы сайта более интерактивными. Это программный код, который встраивается в страницу и срабатывает при определенных событиях.

Скрипт обрабатывается браузером на стороне пользователя. Сервер, с которого загружен сайт, не получает информации о работе скриптов.

На JavaScript создаются такие элементы как выпадающие меню, вращающийся текст, часы, онлайн-калькуляторы и др.

Не следует путать JavaScript и язык программирования Java.

Данное пособие дает лишь базовые представления о работе JavaScript. Для дальнейшего использования рекомендуется изучить справочники, приведенные в конце пособия.

## Основы синтаксиса JavaScript

JavaScript – это язык программирования (ЯП), отдаленно похожий на C, и он обладает основными элементами синтаксиса ЯП.

Как и во многих языках, предложения на языке JavaScript можно разделять точкой с запятой. Но при переходе на новую строку в JavaScript точка с запятой становится необязательна, ее можно не ставить. Следующие строк абсолютно эквивалентны:

a = 10

a = 10;

В отличие от HTML JavaScript чувствителен к регистру символов. Поэтому x и X – это разные переменные.

Однострочные комментарии пишутся после //, многострочные – внутри /\*...\*/.

Присваивание в JavaScript обозначается через =, а не :=. Строки можно писать и в двойных ("текст"), и в одинарных ('текст') кавычках. Объявление переменной:

var Hi = "Hello, World!"

Переменные могут быть глобальными и локальными.

Переменные в JavaScript не типизируются, но им указывается начальное значение. По этому значению и определяется тип данных. Они автоматически преобразуются из одного типа в другой. Иначе говоря, они *слабо типизированы*. Т.е. тип данных есть, но он не задан жестко. Поэтому можно записать:

var x = 10 //целое число

var y = 3.5 //дробное число

var str = "x = " //строка

x = y + x //целое + дробное = дробное (x = 13.5)

str = str + x //строка + число = строка (str = "x = 13.5")

Оператор + в данном случае используется для конкатенации строк.

Знаки арифметических операций:

+ сложение

- вычитание

\* умножение

/ деление

% остаток от деления

Присваивание:

+= прибавить к значению переменной (x += 10 это то же, что и x = x + 10)

-= вычесть из к значения переменной (x -= 10 это то же, что и x = x - 10)

++ прибавить 1 (x++ 1 это то же, что и x = x + 1, и что x += 1)

-- вычесть 1 (x-- 1 это то же, что и x = x – 1, и что x -= 1)

Знаки сравнения:

== равно (в Паскале = )

!= не равно (в Паскале <>)

> больше

>= больше или равно

< меньше

<= меньше или равно

Логические операторы:

&& И (в Паскале and)

|| ИЛИ (в Паскале or)

! НЕ (в Паскале not)

Условный оператор;

if(i>0)

{

...

}

else

{

...

}

Примеры сложных условий:

((k < 5) || (k > 10)) && (x != 0)

!((str == "белый") || (str == "красный"))

Цикл for

for(i=0;i<9;i++)

{

...

}

В скобках указано: i=0 начальное значение счетчика цикла; i<9 до какого значения изменять счетчик; i++ ­на сколько изменять значение счетчика в каждом цикле.

Другой пример, счетчик будет работать от -100 до 100 с шагом 10:

for(i=-100;i<100;i+=10)

Цикл while

while(j<k)

{

...

j++;

}

Массивы задаются конструктором Array, длина массива изменяется динамически

var Arr1 = Array() //пустой масив

var Colors = Array("красный", "синий", "зеленый")

//массив из 3 элементов

Нумерация элементов начинается с 0. Пустые элементы также заполняются нулями

Arr1[3] = 4 //теперь Arr1 = (0,0,0,4)

Массивы в JavaScript содержат встроенные методы

join ­– объединить (добавить элементы в конец)

reverse – развернуть (обратный порядок элементов)

sort – сортировка по возрастанию

var Arr2 = Array("3","30","-5","8")

Arr2 = Array.join("12")

// Arr2 = ("3","30","-5","8", "12")

Arr2 = Array.sort()

// Arr2 = ("-5","3","8","12","30")

Arr2 = Array.reverse()

// Arr2 = ("30","12","8","3","-5")

Объявление функций:

function <имя\_функции>(<параметр1>,<параметр2>,...)

{

return <возвращаемое\_значение>

}

Пример:

function Sum(a, b)

{

return a + b

}

Функция может не иметь параметров и возвращаемого значения, или иметь их несколько:

function IsNegative(x)

{

if (x < 0)

{

return "да"

}

else

{

return "нет"

}

}

function DoNothing()

{

}

## Встраивание JavaScript в HTML-страницу

Встраивание скрипта происходит внутри пар тегов

<SCRIPT lanquaqe ="JavaScript">

//здесь пишется программный код.

</SCRIPT>

Таких блоков на странице может быть сколько угодно и размещаться они могут где угодно. Но поскольку не все браузеры понимают JavaScript, блок скриптов желательно размещать в разделе HEAD, или брать в комментарии HTML.

<SCRIPT lanquaqe ="JavaScript">

<!-- JAVA SCRIPT

//здесь пишется программный код.

-->

</SCRIPT>

Можно вынести скрипты в отдельный файл с расширением js. Подключение файла скриптов выполняется так:

<script src="my\_script.js"></script> /\*подключить my\_script.js, лежащий в той же папке, что и страница\*/

<script src="my/script2.js"></script> /\*подключить script2.js из папки my\*/

## Модель объектов JavaScript

JAVASCRIPT – объектный язык, в нем все переменные и функции являются объектами. У них есть методы и свойства, которые записываются через точку. Выше мы уже рассматривали методы массивов sort, reverse, join.

У строк, например, есть методы toUpperCase и toLowerCase.

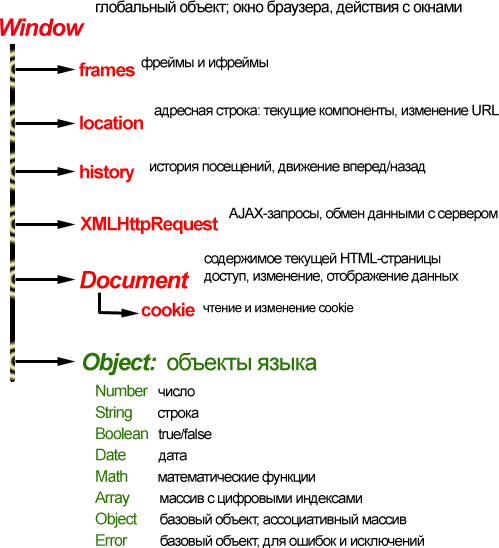
var str1 = "Some text"

var str2 = str1.toUpperCase() //str2 = "SOME TEXT"

var x = 1

str2 x.toUpperCase() //Ошибка! x не является строкой

Кроме того, JAVASCRIPT взаимодействует с различными *глобальными объектами* – окном браузера, веб-страницей, отдельными тегами. В javascript страница представлена в виде объектной модели DOM (Document Object Model).



Любые действия со страницей требуют вызова соответствующего метода DOM. Например, закрытие текущего окна:

window.close()

## События

События возникают при каком-либо действии пользователя на сайте: нажатии на кнопку, наведении кнопки мыши и т.д.

События используются, чтобы запустить JavaScript. Например: вычислить сумму двух полей при нажатии на кнопку. Увеличить картинку при наведении кнопки мыши.

Наиболее часто используемые события:

* *onLoad* – выполнение скрипта или функции при загрузке;
* *onChange* – изменение введенного текста;
* *onClick* – щелчок по объекту;
* *onSelect*– выделение текстового объекта;
* *onUnload* – при загрузке другой страницы;
* *onMouseOver* – при перемещении указателя мыши над объектом.
* *onMouseEnter* – при входе указателя мыши в область объекта.
* *onMouseLeave* – при выходе указателя мыши из области объекта.

События можно привязать к любым объектам.

Например,

<input type=text name=weight onchange="change\_weight()">

при вводе каждого символа будет вызываться функция change\_weight().

Другой пример:

<p>Нажмите <font color=red onclick="show\_pic()">сюда</font>, чтобы увидеть картинку.

При нажатии на красный текст «сюда» будет вызываться функция show\_pic, которая, очевидно, отображает картинку на экране.

# Задания

## Задание 1. Приветствие

Создадим простую страничку:

<HTML>

<HEAD>Задания по JavaScript</HEAD>

<BODY>

<h1>JavaScript. Задание 1</h1>

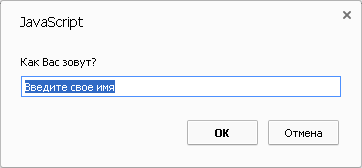
<BODY>

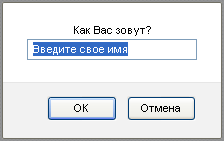
</HTML>

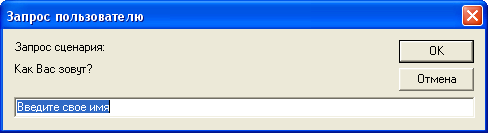
Добавим код, который будет запрашивать имя пользователя и выводить приветствие. Для ввода имени будем использовать окно сообщения prompt:

**prompt**(<Текст запроса>,<Значение по умолчанию>);

Его вид немного отличается в разных браузерах:

  
Google Chrome

  
Firefox

  
Internet Explorer

Вывод окна сообщения осуществляется через функцию alert:

**alert**(<Текст сообщения>);

Сообщение alert выглядит аналогично, только без поля ввода.

В итоге получим скрипт:

<SCRIPT language="JavaScript">

var user\_name = prompt("Как Вас зовут?","Введите свое имя")

alert("Добро пожаловать, " + user\_name + "!")

</SCRIPT>

Изменим страницу, так чтобы приветствие выводилось не в окне, а непосредственно на странице.

Для этого нам понадобится объект, в который будем выводить информацию. Для этого подойдет любой тег с заданным именем. Имя задается через атрибут id. Например, воспользуется тегом <P>

<p id=welcome>Добро пожаловать!</p>

Текст внутри тега задается свойством innerHTML или innerText. Но innerText не работает в Firefox. Поэтому, чтобы вывести приветствие на страницу вместо alert напишем:

welcome.innerHTML = "Добро пожаловать, " + user\_name + "!"

По-другому можно записать:

document.getElementById("welcome").innerHTML = "Добро пожаловать, " + user\_name + "!"

Протестируйте скрипт в двух вариантах. Самостоятельно добавьте проверку – если пользователь не стал вводить имя, выдать сообщение "Добро пожаловать, гость!". Пустая строка обозначается null.

## Задание 2. Перевод градусов Цельсия в шкалу Фаренгейта

Перевод можно осуществить по формуле:

***Градусы Фаренгейта=1.8\* Градусы Цельсия +32;***

Модифицируем скрипт из предыдущего задания:

<SCRIPT language="JavaScript">

var Celsius = prompt("Перевод градусов Цельсия в шкалу Фаренгейта ","Введите градусы Цельсия")

alert("В Фаренгейтах это " + (1.8\*Celsius + 32) )

</SCRIPT>

Однако такая работа скрипта далека от совершенства – он срабатывает только один раз при загрузке страницы, результат нигде не сохраняется.

Для начала, оформим сам перевод в виде функции, пока без параметров:

function C2F()

{

var Celsius = prompt("Перевод градусов Цельсия в шкалу Фаренгейта ","Введите градусы Цельсия")

alert("В Фаренгейтах это " + (1.8\*Celsius + 32) )

}

Теперь добавим на страницу кнопку и пояснение. По нажатию этой кнопки будет вызываться наша функция C2F.

<p>Перевод из градусов Цельсия в Фаренгейты</p>

<input type=button onclick=C2F() value="Перевод">

По аналогии с предыдущим заданием добавим на страницу тег div для вывода результата:

<div id=trans\_result>Здесь будет отображаться результат</div>

Для ввода первоначального значения воспользуемся полем для ввода:

<input type=text id=Celsius value="Введите сюда градусы Цельсия">

Более подробно о поле для ввода, кнопках и других элементах управления см. лабораторную работу Активные формы.

Обратите внимание, мы присвоили полю для ввода имя Celsius. Теперь переменная Celsius в скриптах нам не нужна. Мы будем получать значение непосредственно из свойства value.

function C2F()

{

trans\_result.innerHTML ="В Фаренгейтах это " + (1.8 \* Celsius.value + 32)

}

В качестве финального штриха сделаем так, чтобы после перевода цвет текста изменялся на красный:

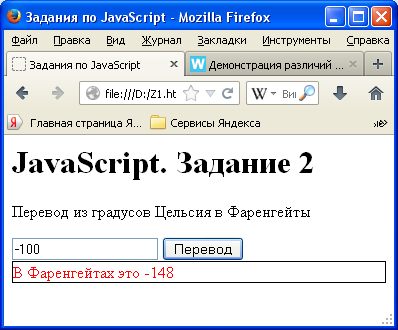
trans\_result.style.color = "Red"

trans\_result.style.border = "solid 1px black"

Через свойсвтво style можно получить доступ к любым настройкам стилей, доступным в CSS (см. соответствующую лабораторную). Того же эффекта можно было добиться одной строкой:

trans\_result.style = "color: Red; border: solid 1px black;"

В результате страница будет выглядеть так:



Протестируйте скрипт в разных браузерах. Самостоятельно измените страницу и скрипт так, чтобы был возможен перевод и в обратную сторону. Пусть при этом оформление тега div становится другим.

## Задание 3. Часы

Достаточно часто на страницах сайта можно увидеть календарь и часы.

Для работы со временем используется специальный объект Date. Изначально он не существует на странице или где-то еще, его необходимо создать конструкцией вида:

var d = new Date

Оператор new выделяет область памяти для заданного типа данных.

Теперь к d можно обращаться как к объекту. В нем содержатся все сведения о текущей дате и времени (на тот момент, когда мы выполнили присваивание):

getYear() **–** год (за 0 считается 1900 год);

getMonth() **-** номер месяца (0 – янв; 1 – фев, ...);

getDate() **-** число;

getDay()- номер дня недели (0 – вс, 1 – пн, ...);

getHours() **-** часы;

getMinutes() **-** минуты;

getSeconds() **-** секунды.

Запишем функцию

function GetTime()

{

var d = new Date();

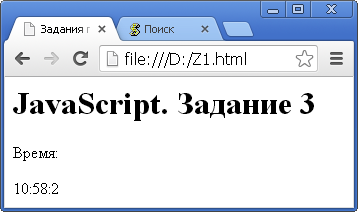
time\_block.innerHTML = d.getHours() + ':' + d.getMinutes() + ':' + d.getSeconds();

}

Очевидно, она будет записывать время в объект с именем time\_block. Самостоятельно добавьте его на страницу, по аналогии с предыдущим заданием.

Но откуда вызывать GetTime? Нам нужно, чтобы время отобразилось сразу, а не после нажатия на кнопку. Для этого воспользуется свойством onload тега body.

Итоговый вид страницы:



В этом примере виден недостаток скрипта: секунды отображаются как «:2», а не «:02». Исправим это:

function GetTime()

{

var d = new Date();

var h = d.getHours();

var m = d.getMinutes();

var s = d.getSeconds();

//часы

if (h > 9)

{

var time\_str = h;

}

else

{

time\_str = "0" + h;

};

//минуты

if (m > 9)

{

time\_str = time\_str + ":" + m;

}

else

{

time\_str = time\_str + ":0" + m;

};

//секунды

if (s > 9)

{

time\_str = time\_str + ":" + s;

}

else

{

time\_str = time\_str + ":0" + s;

};

time\_block.innerHTML = time\_str;

}

Этот код довольно длинный и состоит из однотипных кусков. Как его можно упростить? Одинаковые куски кода желательно всегда выносить в отдельные функции. Для этого они и существуют.

function PrintWith0(x)

{

if (x > 9)

{

return x;

}

else

{

return "0" + x;

};

}

function GetTime()

{

var d = new Date();

var h = d.getHours();

var m = d.getMinutes();

var s = d.getSeconds();

var time\_str = PrintWith0(h) + ":" + PrintWith0(m) + ":" + PrintWith0(s);

time\_block.innerHTML = time\_str;

}

Теперь заставим наши часы «тикать», т.е. обновлять время каждую секунду. Для этого используется функция setTimeout:

setTimeout(<скрипт\_для\_вызова>,<задержка\_в\_мс>)

Добавим в конец функции GetTime() строчку:

setTimeout("GetTime()",1000);

Т.е. GetTime будет вызывать саму себя, но с задержкой в 1000мс (1с). Теперь время будет постоянно обновляться. При этом сама страница перезагружаться не будет.

Добавим оформление стиля для часов. Вынесите его в отдельный css-файл

.clock

{

font-family: Arial;

font-size: 16pt;

background-color: black;

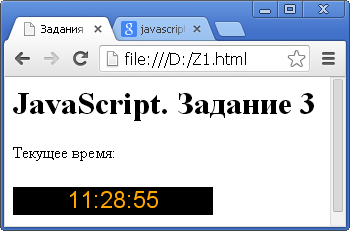
color: orange;

width: 200px;

text-align: center;

border-width: 1pt;

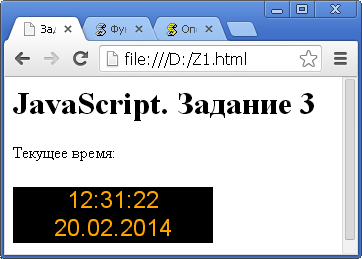
}



По аналогии отобразим текущую дату:

var date\_str = PrintWith0(d.getDate()) + "." + PrintWith0(d.getMonth() + 1) + "." + PrintWith0(d.getYear() + 1900);

time\_block.innerHTML = "<p>" + time\_str + "<br>" + date\_str + "</p>";



Проверьте, будут ли работать часы, если свойство onload указать непосредственно для time\_block.

Добавьте в CSS разные стили оформления часов для ночи, утра, дня и вечера. Переключайте их в зависимости от текущего времени (атрибут class соответствует свойству className в JavaScript).

## Задание 4. Бегущая строка

Создадим строку текста, которая будет перемещаться вдоль нижнего края экрана. Для этого нам потребуется совместное использование CSS и JavaScript.

Бегущую строку мы разместим в двух вложенных тегах div: один будет служить фоном, а другой текстом.

Создайте страницу со следующим содержимым:

<BODY>

<h1>JavaScript. Задание 4</h1>

<div class=scroll\_back id=myscrollback

<div class=scroll id=myscroll>Бегущая строка</div>

</div>

<BODY>

и таблицу CSS для нее:

.scroll

{

position: absolute;

left: 0px;

top: 0px;

font-family: Arial;

font-size: 10pt;

color: yellow;

white-space: nowrap;

}

.scroll\_back

{

position: absolute;

left: 0px;

height: 20px;

width: 100%;

background-color: black;

overflow: hidden;

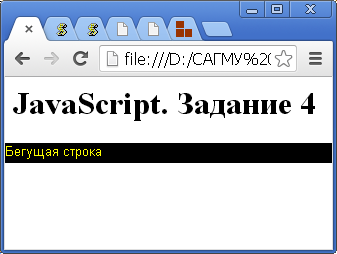
}

Обратите внимание на свойства position: absolute, white-space: nowrap и overflow: hidden.

position: absolute означает, что мы будем задавать положение наших элементов в абсолютных единицах: scroll\_back относительно всего окна, а scroll относительно scroll\_back.

white-space: nowrap показывает, что текст **не** нужно переносить, даже если он не умещается в одну строку.

overflow: hidden значит, что не умещающийся текст следует обрезать.



Пока что наша бегущая строка не двигается и располагается вверху страницы. Размещение текста будет выполнять JavaScript.

Сначала добавим функцию для организации прокрутки текста.

var pos = 0;//глобальная переменная для хранения текущего положения начала строки

function ScrollText()

{

//если pos не вылезает за границы myscrollback

if (pos < myscrollback.offsetWidth)

{

pos++ //увеличиваем pos на 1

}

else

{

//иначе – перемещаем строку в начало, за левый край страницы

pos = -myscroll.offsetWidth

}

//собственно перемещение текста

myscroll.style.left = pos + "px";

//перезапуск каждые 20мс (50fps)

setTimeout("ScrollText()",20);

}

Обратите внимание – ширину элемента можно узнать с помощью свойства offsetWidth, а высоту, соответственно через offsetHeight.

Добавим функцию, которая «прилепит» бегущую строку к нижнему краю листа:

function BottomText()

{

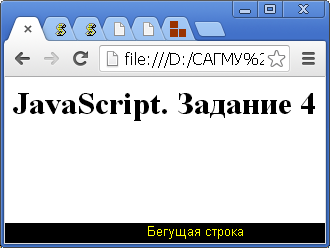
myscrollback.style.top = (window.innerHeight - 20) + "px"

}

Обе их будем вызывать по onload:

<BODY onload="ScrollText(); BottomText()">

Теперь бегущая строка работает.



Но если вы попытаетесь изменить размер окна браузера, то заметите, что бегущая строка немного «не успевает» за окном.

Чтобы этого избежать добавим вызов BottomText еще и при изменении размера окна:

<BODY onload="ScrollText(); BottomText()" onresize="BottomText()">

Замечание: это учебный пример. Для бегущей строки существует специальный тег <marquee>. Но его возможности более ограничены.

Самостоятельно измените скрипт так, чтобы цвет бегущей строки менялся после каждого прокручивания.

Измените скрипт так, чтобы добежав до правого края, строка начинала движение в обратную сторону.

## Задание 5. Галерея

Создадим галерею, в которой будут отображаться уменьшенные копии картинок, причем при клике на картинку она отобразится в полном размере в отдельном поле, а при наведении мыши – немного увеличится.

На странице разместим уменьшенные копии рисунков в виде горизонтальной таблицы и под таблицей – текущую картинку.



Исходный код страницы:

<h1>JavaScript. Задание 5</h1>

<p>Галерея:</p>

<table height=120>

<tr valign=center>

<td><img src="15-1461.jpg" height=70>

</td>

<td><img src="018052.jpg" height=70>

<td>

<td><img src="69049.jpg" height=70>

</td>

<td><img src="144066.jpg" height=70>

</td>

<td><img src="454044.jpg" height=70>

</td>

</tr>

</table>

<p align=center><img id=big\_pic src="15-1461.jpg"></p>

Все картинки в галерее отображаются с одинаковой высотой (70), независимо от реального размера. В big\_pic будет отображаться картинка в реальном размере.

Для выполнения задания нам потребуется три функции:

1. для загрузки большой картинки (onClick).
2. для увеличения картинки в галерее (onMouseEnter).
3. для обратного уменьшения картинки в галерее (onMouseLeave).

Имя загружаемой картинки будем передавать в функцию в виде параметра:

function ShowPic(filename)

{

big\_pic.src = filename

}

Функцию будем вызывать по событию onclick маленьких картинок.

onclick="ShowPic('15-1461.jpg')"

Текстовый параметр передаем в одинарных кавычках.

Для уменьшения и увеличения картинки нам нужно знать, какую картинку увеличивать. Саму картинку также можно передать в качестве параметра.

function ZoomPic(pic)

{

pic.height = 100

}

function UnZoomPic(pic)

{

pic.height = 70

}

Но как ее идентифицировать? Можно задать каждой маленькой картинке имя (id) и передавать его, например для первой картинки:

<img id=pic1 src="15-1461.jpg" height=70

onclick="ShowPic('15-1461.jpg')"

onmouseenter="ZoomPic(pic1)"

onmouseleave="UnZoomPic(pic1)">

Но это не очень удобно, особенно, когда картинок много.

В JavaScript используется указатель на текущий объект this (аналог self в Дельфи).

<img src="15-1461.jpg" height=70

onclick="ShowPic('15-1461.jpg')"

onmouseenter="ZoomPic(this)"

onmouseleave="UnZoomPic(this)">

this – это тот объект, с которым произошло событие.

Самостоятельно измените код так, чтобы в ShowPic автоматически передавался текущий рисунок (т.е. не приходилось вручную повторять имя рисунка в параметрах). При клике выведите на экран не только сам рисунок, но и имя файла в виде ссылки.

*Замечание*: для экономии трафика и ускорения загрузки страницы правильнее хранить уменьшенные картинки в виде отдельных файлов. А уже когда пользователь нажал на картинку, загружать ее полный вариант.

## Рекомендуемые источники

1. <http://javascript.ru/>
2. <http://uroki-javascript.ru/>
3. Справочник JavaScript <https://developer.mozilla.org/ru/docs/JavaScript>