

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ»

Д.В. Груздев

ПРАКТИКА ЭВМ – JAVASCRIPT (3 КУРС)

Учебное пособие

Воронеж
Издательский дом ВГУ
2017

JavaScript

JavaScript изначально создавался для того, чтобы сделать web-странички «живыми». Программы на этом языке называются *скриптами*. Они подключаются напрямую к HTML и, как только загружается страничка — тут же выполняются.

Программы на JavaScript — обычный текст. Они не требуют какой-то специальной подготовки.

В этом плане JavaScript сильно отличается от другого языка, который называется Java.

Почему JavaScript?

Когда создавался язык JavaScript, у него изначально было другое название: «LiveScript». Но тогда был очень популярен язык Java, и маркетинологи решили, что схожее название сделает новый язык более популярным.

Планировалось, что JavaScript будет эдаким «младшим братом» Java. Однако, история распорядилась по-своему, JavaScript сильно вырос, и сейчас это совершенно независимый язык, со своей спецификацией, которая называется [ECMAScript](#), и к Java не имеет никакого отношения.

У него много особенностей, которые усложняют освоение, но по ходу учебника мы с ними разберемся.

Чтобы читать и выполнять текст на JavaScript, нужна специальная программа — [интерпретатор](#). Процесс выполнения скрипта называют «*интерпретацией*».

Компиляция и интерпретация, для программистов

Строго говоря, для выполнения программ существуют «компиляторы» и «интерпретаторы».

Компиляторы преобразуют программу в машинный код. Этот машинный код затем распространяется и запускается.

А интерпретаторы, в частности, встроенный JS-интерпретатор браузера — получают программу в виде исходного кода. При этом распространяется именно сам исходный код (скрипт).

Современные интерпретаторы перед выполнением преобразуют JavaScript в машинный код или близко к нему, а уже затем выполняют.

Во все основные браузеры встроен интерпретатор JavaScript, именно поэтому они могут выполнять скрипты на странице.

Но, разумеется, этим возможности JavaScript не ограничены. Это полноценный язык, программы на котором можно запускать и на сервере, и даже в стиральной машинке, если в ней установлен соответствующий интерпретатор.

Что умеет JavaScript?

Современный JavaScript — это «безопасный» язык программирования общего назначения. Он не предоставляет низкоуровневых средств работы с памятью, процессором, так как изначально был ориентирован на браузеры, в которых это не требуется.

В браузере JavaScript умеет делать все, что относится к манипуляции со страницей, взаимодействию с посетителем и, в какой-то мере, с сервером:

- Создавать новые HTML-теги, удалять существующие, менять стили элементов, прятать, показывать элементы и т.п.
- Реагировать на действия посетителя, обрабатывать клики мыши, перемещение курсора, нажатие на клавиатуру и т.п.
- Посылать запросы на сервер и загружать данные без перезагрузки страницы (эта технология называется "AJAX").

В новых версиях JavaScript (ECMAScript) эти недостатки постепенно убирают. Процесс внедрения небыстрый, в первую очередь из-за старых версий IE, но они постепенно вымирают. Современный IE в этом отношении несравнимо лучше.

Тег SCRIPT

Программы на языке JavaScript можно вставить в любое место HTML при помощи тега SCRIPT. Например:

```
<!DOCTYPE HTML>
<html>
<head>
  <!-- Тег meta для указания кодировки -->
  <meta charset="utf-8">
</head>
<body>

  <p>Начало документа...</p>

  <script>
    alert('Привет, Мир!');
  </script>

  <p>...Конец документа</p>

</body>
</html>
```

Этот пример использует следующие элементы:

<script> ... </script>

Тег script содержит исполняемый код. Предыдущие стандарты HTML требовали обязательного указания атрибута type, но сейчас он уже не нужен. Достаточно просто <script>.

Браузер, когда видит <script>:

- Начинает отображать страницу, показывает часть документа до script
- Встретив тег script, переключается в JavaScript-режим и не показывает, а исполняет его содержимое.
- Закончив выполнение, возвращается обратно в HTML-режим и отображает оставшуюся часть документа.

Попробуйте этот пример в действии, обратите внимание что **пока браузер не выполнит скрипт - он не может отобразить часть страницы после него.**

alert(...)

Отображает окно с сообщением и ждет, пока посетитель не нажмет «Ок»

Кодировка и тег META

При попытке сделать такой же файл у себя на диске и запустить, вы можете столкнуться с проблемой — выводятся «кракозяблы», «квадратики» и «вопросики» вместо русского текста.

Чтобы всё было хорошо, нужно:

- Убедиться, что в HEAD есть строка <meta charset="utf-8">. Если вы будете открывать файл с диска, то именно он укажет браузеру кодировку.
- Убедиться, что редактор сохранил файл в кодировке UTF-8, а не, скажем, в windows-1251. На

английском соответствующий параметр может называться «charset» или «encoding». Указание кодировки — часть обычного HTML, к JavaScript не имеет отношения.

Очень важно не только читать, но и тестировать, пробовать писать что-то самому. ^[1]_{SEP}Решите задачу, чтобы удостовериться, что вы все правильно поняли.

Современная разметка для тега SCRIPT

В старых скриптах оформление тега SCRIPT было немного сложнее. В них можно встретить следующие элементы:

Атрибут `<script type=...>`

В отличие от HTML5, стандарт HTML 4 требовал обязательного указания этого атрибута. Выглядел он так: `type="text/javascript"`.

Если вы укажете некорректные данные в атрибуте `type`, например `<script type="text/html">`, то содержимое тега не будет отображено. Но его можно получить средствами JavaScript. Этот хитрый способ используют для добавления служебной информации на страницу.

Атрибут `<script language=...>`

Этот атрибут ставить не обязательно, т.к. язык по умолчанию — JavaScript.

Комментарии до и после скриптов

В старых руководствах и книгах иногда рекомендуют использовать HTML-комментарии внутри SCRIPT, чтобы спрятать JavaScript от браузеров, которые не поддерживают его.

Выглядит это примерно так:

```
<script type="text/javascript"><!--
...
//--></script>
```

Браузер, для которого предназначались такие трюки, очень старый Netscape, давно умер. Поэтому в этих комментариях нет нужды.

Внешние скрипты

Если JavaScript-кода много — его выносят в отдельный файл, который подключается в HTML:

```
<script src="/path/to/script.js"></script>
```

Здесь `/path/to/script.js` - это абсолютный путь к файлу, содержащему скрипт (из корня сайта).

Браузер сам скачает скрипт и выполнит.

Например:

```
<html>
<head>
  <meta charset="utf-8">
  <script src="/files/tutorial/browser/script/rabbits.js"></script>
</head>

<body>
  <script>
    count_rabbits();
  </script>
</body>

</html>
```

Содержимое файла /files/tutorial/browser/script/rabbits.js:

```
function count_rabbits() {
  for(var i=1; i<=3; i++) {
    alert("Кролик номер "+i)
  }
}
```

Можно указать и полный URL, например:

```
<script src="http://code.jquery.com/jquery.js"></script>
```

Вы также можете использовать путь относительно текущей страницы, например src="script.js" если скрипт находится в том же каталоге, что и страница.

Чтобы подключить несколько скриптов, используйте несколько тегов:

```
<script src="/js/script1.js"></script>
<script src="/js/script2.js"></script>
```

...

Как правило, в HTML пишут только самые простые скрипты, а сложные выносят в отдельный файл.

Благодаря этому один и тот же скрипт, например, меню или библиотека функций, может использоваться на разных страницах.

Браузер скачает его только первый раз и в дальнейшем, при правильной настройке сервера, будет брать из своего [кэша](#).

Если указан атрибут src, то содержимое тега игнорируется.

В одном теге SCRIPT нельзя одновременно подключить внешний скрипт и указать код.

Вот так не сработает:

```
<script src="file.js">
  alert(1); // если указан src, то внутренняя часть скрипта игнорируется
</script>
```

Нужно выбрать: либо SCRIPT идёт с src, либо содержит код. Тег выше следует разбить на два: один — с src, другой с кодом:

```
<script src="file.js"></script>
<script>
  alert(1);
</script>
```

Команды

Например, можно вместо одного вызова alert сделать два:

```
alert('Привет'); alert('Мир');
```

Комментарии

Со временем программа становится большой и сложной. Появляется необходимость добавить *комментарии*, которые объясняют, что происходит и почему.

Комментарии могут находиться в любом месте программы и никак не влияют на ее выполнение. Интерпретатор JavaScript попросту игнорирует их.

Однотрочные комментарии начинаются с двойного слэша //. Текст считается комментарием до конца строки:

```
// Команда ниже говорит "Привет"
alert('Привет');
```

Переменная

Переменная состоит из имени и выделенной области памяти, которая ему соответствует.

Для объявления или, другими словами, создания переменной используется ключевое слово var:

```
var message;
```

После объявления, можно записать в переменную данные:

```
var message;
message = 'Привет'; // сохраним в переменной строку
```

Эти данные будут сохранены в соответствующей области памяти и в дальнейшем доступны при обращении по имени:

```
var message;
message = 'Привет';

alert(message); // выведет содержимое переменной
```

Важность директивы var

В JavaScript вы можете создать переменную и без var, достаточно просто присвоить ей значение:

```
x = "value"; // переменная создана, если ее не было
```

Технически, это не вызовет ошибки, но делать так все-таки не стоит.

Всегда определяйте переменные через var. Это хороший тон в программировании и помогает избежать ошибок.

```
<html>
<head>
  <meta http-equiv="X-UA-Compatible" content="IE=8">
</head>
<body>
  <div id="test"></div>

  <script>
    test = 5;
    alert(test);
  </script>

</body>
</html>
```

Всё будет хорошо, если объявить test, используя var:^[1]^[2]Правильный код:

```
<html>
<body>
  <div id="test"></div>

  <script>
    var test = 5;
    alert(test);
  </script>

</body>
</html>
```

Константы

Константа — это переменная, которая никогда не меняется. Как правило, их называют большими буквами, через подчёркивание. Например:

```
var COLOR_RED = "#F00";
var COLOR_GREEN = "#0F0";
var COLOR_BLUE = "#00F";
var COLOR_ORANGE = "#FF7F00";
```

```
alert(COLOR_RED); // #F00
```

Технически, константа является обычной переменной, то есть её можно изменить. Но мы договариваемся этого не делать.

Зачем нужны константы? Почему бы просто не использовать "#F00" или "#0F0"?

- Во-первых, константа — это понятное имя, в отличие от строки "#FF7F00".
- Во-вторых, опечатка в строке может быть не замечена, а в имени константы её упустить невозможно — будет ошибка при выполнении.

Константы используют вместо строк и цифр, чтобы сделать программу понятнее и избежать ошибок.

Имена переменных

На имя переменной в JavaScript наложены всего два ограничения.

Имя может состоять из: букв, цифр, символов \$ и _

Первый символ не должен быть цифрой.

Например:

```
var myName;
var test123;
```

Что здесь особенно интересно - доллар '\$' и знак подчеркивания '_' являются такими же обычными символами, как буквы:

```
var $ = 5; // объявили переменную с именем '$'
var _ = 15; // переменная с именем '_'

alert($);
```