**Основы HTML**

Встроенная в браузер JavaScript-консоль, которой мы до сих пор пользовались, хороша, когда нужно протестировать небольшой фрагмент кода, но для создания более масштабных программ понадобится чуть более гибкое и универсальное средство — вроде веб-страницы со встроенным JavaScript-кодом. В этой главе мы как раз и научимся создавать несложные странички на языке HTML.

Гипертекстовый язык разметки HTML предназначен специально для создания веб-страниц. Слово гипертекстовый означает, что фрагменты текста связаны между собой гиперссылками — то есть ссылками в документе на другие объекты. А язык разметки — это способ встраивать в текст дополнительную информацию. Разметка указывает программам (таким как браузер), как отображать текст и что с ним делать.

Сейчас давайте попробуем научиться создавать HTML-документы в текстовом редакторе — программе, предназначенной для работы с простым текстом без форматирования, в отличие от текстовых процессоров вроде Microsoft Word. Документы текстовых процессоров содержат форматированный текст (с различными типами и размерами шрифтов, цветами и т. п.), и устроены эти программы так, чтобы форматирование было легко менять. Кроме того, многие текстовые процессоры позволяют вставлять в текст картинки и другие графические элементы.

Простой же текст является только текстом — без цветов, стилей, размеров и т. д. Вставить в такой текст картинку не выйдет, разве только составить ее из символов.

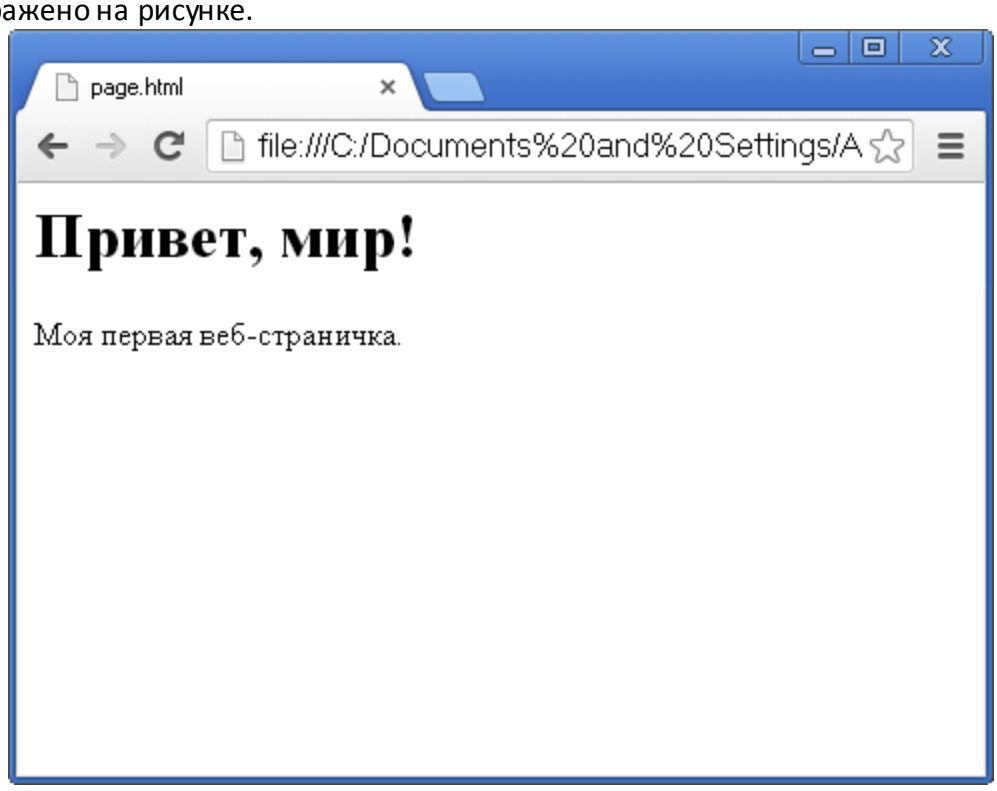
**Наш первый HTML-документ**

Запустите Блокнот и затем выберите Файл ► Сохранить, чтобы сохранить новый, пустой файл; назовите его page.html и сохраните, например, на рабочий стол.

Настало время писать HTML-код. Введите в файл page.html следующий текст:

```
<meta charset="UTF-8">
<h1>Привет, мир!</h1>
<p>Моя первая веб-страничка.</p>
```

Сохраните обновленный файл page.html, выбрав Файл ► Сохранить. Теперь посмотрим, на что это будет похоже в веб-браузере. Откройте Chrome и, удерживая CTRL, нажмите O. В появившемся окне выберите файл page.html, находящийся на рабочем столе. То, что вы должны после этого увидеть, изображено на рисунке.




**Теги и элементы**

HTML-документы состоят из элементов. Каждый элемент начинается с открывающего тега и оканчивается закрывающим тегом. Например, в нашем первом документе пока всего два элемента: h1 и p (а также элемент meta, но его мы отдельно здесь рассматривать не будем. Он

нужен, чтобы в браузере отображался русский текст). Элемент h1 начинается с открывающего тега <h1> и заканчивается закрывающим тегом </h1>, а элемент p начинается с открывающего тега  и заканчивается закрывающим тегом . Все, что находится между открывающим и закрывающим тегами, называют содержимым элемента.

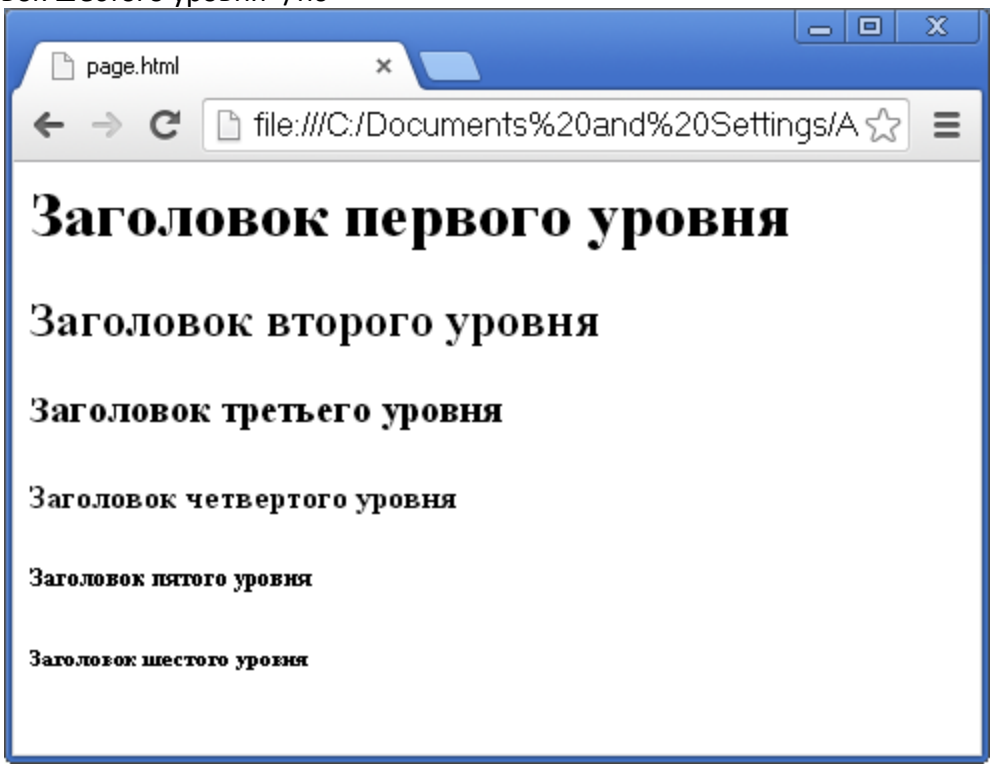
Открывающие теги представляют собой название элемента в угловых скобках: < и >. Закрывающие теги выглядят также, но перед именем элемента в них ставится наклонная черта (/).

Элементы заголовков

У каждого элемента есть особое назначение и способ применения. Например, элемент h1 означает «это заголовок верхнего уровня». Содержимое, которое вы введете между открывающим и закрывающим тегами <h1>, браузер отобразит на отдельной строке крупным жирным шрифтом.

Всего в HTML шесть уровней заголовков: h1, h2, h3, h4, h5 и h6. Выглядят они так:

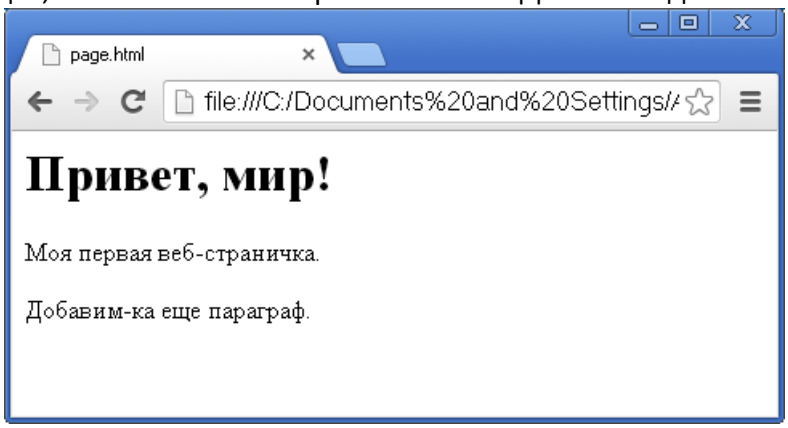
```
<h1>Заголовок первого уровня</h1>
<h2>Заголовок второго уровня</h2>
<h3>Заголовок третьего уровня</h3>
<h4>Заголовок четвертого уровня</h4>
<h5>Заголовок пятого уровня</h5>
<h6>Заголовок шестого уровня</h6>
```




**Элемент p**

Элемент p нужен для разделения текста на параграфы. Любой фрагмент текста, который вы поместите между тегами , будет отображен как отдельный параграф, с отступами сверху и снизу. Давайте посмотрим, что происходит, если элементов  несколько. Для этого добавьте новую строку в документ page.html.

```
<meta charset="UTF-8">
<h1>Привет, мир!</h1>
<p>Моя первая веб-страничка.</p>
<p>Добавим-ка еще параграф.</p>
```

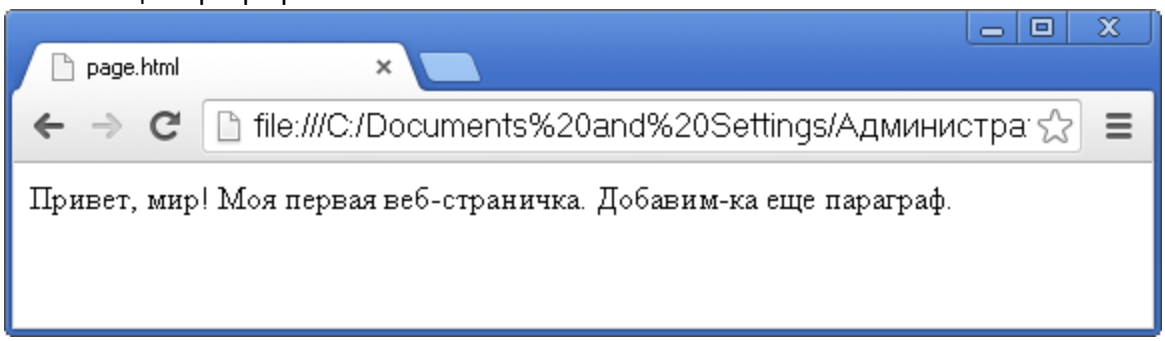



Обратите внимание, что каждый параграф отображен с новой строки, а между параграфами сделан отступ. Все это благодаря тегу .

**Пробелы в HTML и блочные элементы**

А как наша страничка будет выглядеть без тегов? Давайте посмотрим:

```
<meta charset="UTF-8">
Привет, мир!
Моя первая веб-страничка.
Добавим-ка еще параграф.
```



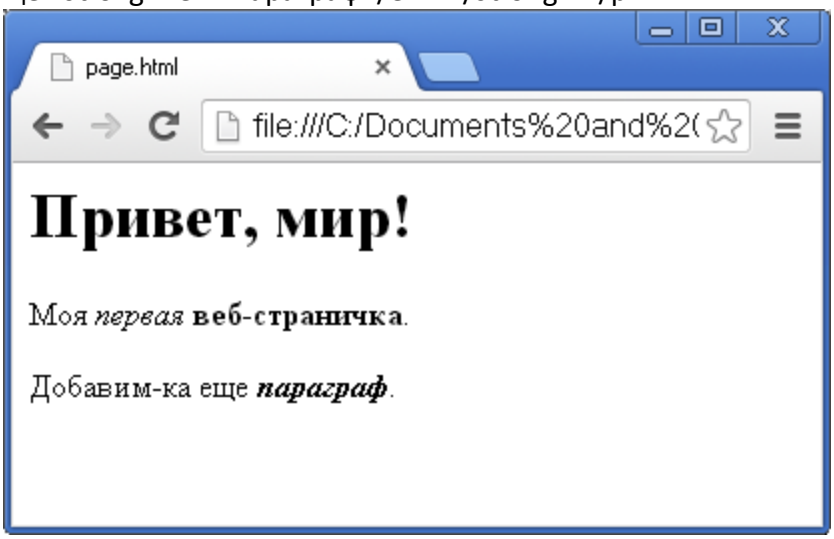

Мало того что пропало форматирование, теперь весь текст отображается в одну строку! Дело в том, что в HTML все пробельные символы преобразуются в единственный пробел. Пробельные символы — это любые символы, которые отображаются в браузере как пробелы или отступы, — например, это пробел, символ табуляции и символ перевода строки (тот самый, который вы вводите, нажимая ENTER или RETURN). Поэтому все пустые строки, которые вы вставите между фрагментами текста в HTML-документе, сожмутся до одного пробела.

Элементы p и h1 — блочные; это значит, что их содержимое отображается отдельными блоками текста с новой строки и любое содержимое, идущее после такого блока, тоже начнется с новой строки.

**Строчные элементы**

А теперь добавим к нашему документу еще два элемента, em и strong:

```
<meta charset="UTF-8">
<h1>Привет, мир!</h1>
<p>Моя <em>первая</em> <strong>веб-страничка</strong>.</p>
<p>Добавим-ка еще <strong><em>параграф</em></strong>.</p>
```




Элемент em отображает свое содержимое курсивом, а элемент strong — жирным шрифтом. И em, и strong относятся к строчным элементам, поскольку они, в отличие от блочных элементов, не выводят свое содержимое отдельной строкой.

Чтобы отобразить текст одновременно жирным шрифтом и курсивом, поместите его внутрь обоих тегов. Обратите внимание, что в последнем примере теги стояли в такой последовательности: параграф. Очень важно правильным образом вкладывать элементы друг в друга: если один элемент находится внутри другого элемента, то его открывающий тег и его закрывающий тег также должны находиться внутри этого элемента. Например, такой вариант недопустим:

```
<strong ><em>параграф</strong></em>
```

Закрывающий тег  расположен здесь перед закрывающим тегом . Как правило, браузеры никак не сообщают о подобных ошибках, однако неправильно вложенные теги приведут к неверному отображению страниц.

**Полноценный HTML-документ**

До сих пор мы имели дело лишь с фрагментами HTML, тогда как полноценный HTML-документ должен включать некоторые дополнительные элементы. Давайте посмотрим на законченный HTML-документ и разберемся, зачем нужна каждая его часть. Добавьте в файл page.html следующие элементы:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Моя первая настоящая HTML-страничка</title>
</head>
<body>
<h1>Привет, мир!</h1>
<p>Моя <em>первая</em> <strong>веб-страничка</strong>.</p>
<p>Добавим-ка еще <strong><em>параграф</em></strong>.</p>
</body>
</html>
```

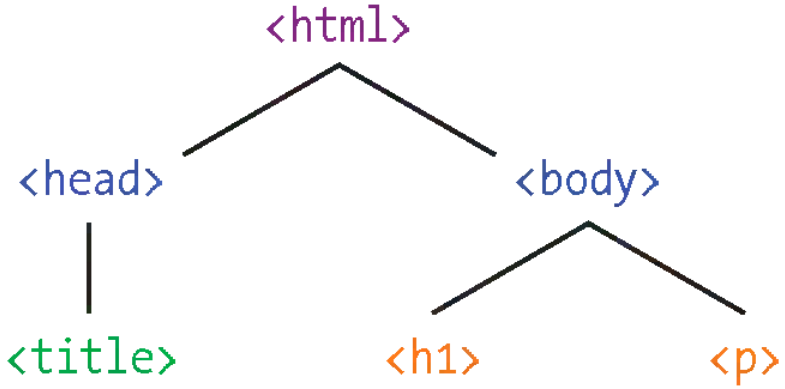
Давайте по очереди рассмотрим элементы из файла page.html. Тег <!DOCTYPE html> — всего лишь объявление, он сообщает: «это HTML-документ». Далее следует открывающий тег <html> (закрывающий тег </html> находится в самом конце кода). Каждый HTML-документ должен содержать элемент html верхнего уровня вложенности.

Внутри элемента html находятся элементы head и body. Элемент head содержит определенную информацию об HTML-документе, например элемент title, устанавливающий название документа, — обратите внимание, что текст на закладке браузера соответствует содержимому title. Элемент title находится внутри элемента head, который, в свою очередь, находится внутри элемента html.

Внутри элемента body находится содержимое, которое отображается в браузере. В данном случае мы просто скопировали эти данные из предыдущего примера.

**Иерархия HTML**

HTML-элементы подчинены строгой иерархии, которую можно себе представить в виде перевернутого дерева.



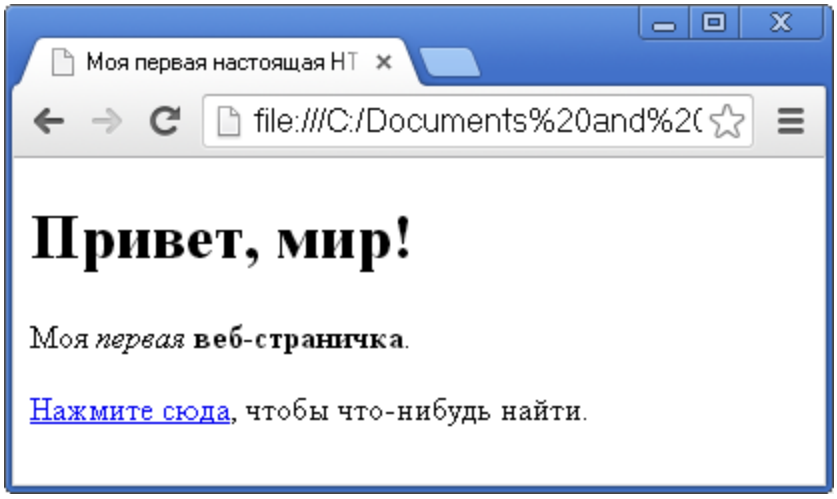

Сверху находится элемент html. Он содержит элементы head и body. В свою очередь, head содержит элемент title, а body — элементы h1 и p. Браузер интерпретирует наш HTML согласно этой иерархии. О том, как менять структуру документа, мы узнаем позже.

**Добавим в HTML ссылки**

Ранее мы узнали, что HTML — гипертекстовый язык. Это значит, что HTML-документы могут содержать гиперссылки (или просто ссылки), ведущие на другие веб-страницы. Такие ссылки можно создавать с помощью элемента a (от английского anchor — «якорь»).

Измените свой HTML-документ, чтобы он соответствовал следующему примеру: удалите второй элемент p, а также теги em и strong и добавьте код, чтобы создать ссылку на интернет-адрес http://yandex.ru:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Моя первая настоящая HTML-страничка</title>
</head>
<body>
<h1>Привет, мир!</h1>
<p>Моя <em>первая</em> <strong>веб-страничка</strong>.</p>
<p><a href="http://yandex.ru">Нажмите сюда</a>, чтобы что-нибудь найти.</p>
</body>
</html>
```




**Атрибуты ссылок**

Давайте разберемся, как мы создали эту HTML-ссылку. Чтобы браузер знал, куда перейти по клику, мы добавили элементу a так называемый атрибут. Атрибуты HTML-документов напоминают пары «ключ-значение» в объектах JavaScript: у каждого атрибута есть имя и значение. Посмотрите еще раз на созданную нами ссылку:

```
<a href="http://yandex.ru">Нажмите сюда</a>
```

В данном случае у атрибута есть имя href и значение "http://yandex.ru" —то есть веб-адрес. Ссылка отправит вас по любому адресу, который указан в качестве значения атрибута href.

Также к ссылкам можно добавлять атрибут title — он задает текст, который появляется при наведении курсора на ссылку. Например, давайте изменим открывающий тег <a>, чтобы он выглядел так:

```
<p><a href="http://yandex.ru" title="Поисковая система">Нажмите сюда</a>
```

Теперь перезагрузите страничку. При наведении мышки на ссылку должна появиться надпись: «Поисковая система».

**Условия и циклы**

Условные конструкции и циклы — одни из самых важных понятий в JavaScript. Условная конструкция представляет собой команду: «если что-то истинно (true), сделай это, иначе сделай то». Пример: выполнив домашнее задание, вы можете съесть мороженое, но, если домашнее задание не готово, мороженое вам не светит. А цикл — это инструкция: «до тех пор, пока что-то истинно (true), продолжай делать это». Пример: пока вы испытываете жажду, продолжайте пить воду.

Условные конструкции и циклы — понятия, лежащие в основе любой мало-мальски серьезной программы. Их называют управляющими конструкциями, поскольку они позволяют решать, какие части кода и когда выполнять, а также насколько часто это нужно делать, исходя из заданных вами условий.

Для начала давайте разберемся, как встраивать JavaScript в HTML-файл. Это позволит писать программы более сложные, чем те, с которыми мы имели дело до сих пор.

**Внедрение JavaScript-кода в HTML**

Вот HTML-файл, который мы создали ранее, с некоторыми дополнениями.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Моя первая настоящая HTML-страничка</title>
</head>
<body>
<h1>Привет, мир!</h1>
<p>Моя первая веб-страничка.</p>
<script>
var message = "Привет, мир!";
console.log(message);
</script>
</body>
</html>
```

Как видите, мы добавили новый элемент под названием script. Этот элемент особенный: содержимое большинства элементов HTML отображается на страничке, однако то, что находится внутри тегов script, считается JavaScript-кодом и передается для выполнения интерпретатору JavaScript. Теперь рассмотрим код внутри элемента script:

```
var message = "Привет, мир!";
console.log(message);
```

Выполнение JavaScript, встроенного в HTML, заметно отличается от запуска кода в консоли. Введенный в консоли код выполнится при первом же нажатии ENTER, после чего вы увидите результат сработавшей команды. Однако код, встроенный в веб-страницу, выполняется сразу и целиком, от верхних строк к нижним, причем в консоль ничего автоматически не выводится — если мы не попросим браузер об этом отдельно. Для вывода в консоль можно воспользоваться командой console.log — это поможет следить за ходом выполнения программы. Метод console.log принимает любое значение и печатает (логирует) это значение в консоли. Например, загрузив в браузер наш последний пример, вы увидите в консоли вот что (разумеется, если она открыта):

```
Привет, мир!
```

Вызов console.log (message) привел к появлению в консоли строки "Привет, мир!".

Теперь, когда вы знаете, как с удобством писать длинные JavaScript-программы, можно перейти к изучению условных конструкций.

**Условные конструкции**

В JavaScript есть два вида условных конструкций — это if и if... else. Оператор if выполняет фрагмент кода, если какое-то условие истинно (true). Например: если вы хорошо себя вели, то получите чебурек. А оператор if... else выполняет один фрагмент кода, если условие дает true, и

другой фрагмент в противном случае. Например: если вы хорошо себя вели, получите чебурек, иначе вам не дадут денег на оплату сотовой связи.

**Конструкция if**

Самая простая из управляющих конструкций JavaScript — это if. Она используется, чтобы запускать код, если некое условие истинно (true). Вернитесь к нашему HTML-файлу и замените содержимое элемента script следующими строками:

```
var surname = "Череззаборногузакиденко";
console.log("Привет, " + surname);
if(surname.length > 12)
{
console.log("Ну и длиннющая же у вас фамилия!");
}
```

Сначала мы создали переменную surname и присвоили ей значение — строку "Череззаборногузакиденко". Затем мы с помощью console.log напечатали строку "Привет, Череззаборногузакиденко".

Далее мы использовали конструкцию if, чтобы проверить: длина surname больше, чем двенадцать символов? Если это так, мы посредством console.log выводим: "Ну и длиннющая же у вас фамилия!".

Конструкция if состоит из двух частей: условия и тела. Условие должно давать булево значение. А тело — одна или несколько строк JavaScript-кода, которые будут выполнены, если условие истинно (true).

После загрузки нашей HTML-странички со встроенным JavaScript-кодом в консоли должно появиться:

```
Привет, Череззаборногузакиденко
Ну и длиннющая же у вас фамилия!
```

В фамилии Череззаборногузакиденко 23 буквы, поэтому surname.length вернет значение 23, и условие surname.length > 12 даст true. В результате будет выполнено тело оператора if, и в консоли появится несколько фамильярное сообщение. Чтобы избежать выполнения if, поменяйте фамилию Череззаборногузакиденко на Николаенко (оставив остальной код без изменений):

```
var surname = "Николаенко";
```

Теперь сохраните файл и перегрузите страничку. На этот раз условие surname.length > 12 даст false, поскольку surname.length равно 10. В итоге тело оператора if выполнено не будет, а в консоли появится лишь:

```
Привет, Николаенко
```

Тело оператора if выполняется, только когда условие дает true. Если же условие дает false, интерпретатор игнорирует конструкцию if и переходит к следующей за ней строке.

**Конструкция if... else**

Как я уже говорил, оператор if запускает код своего тела, только если условие дает true. Но если вы хотите, чтобы по условию false тоже что-то происходило, вам нужна конструкция if... else.

Давайте дополним предыдущий пример:

```
var surname = "Николаенко";
console.log("Привет, " + surname);
if(surname.length > 12)
{
console.log("Ну и длиннющая же у вас фамилия!");
}
else
{
console.log("Фамилия у вас не из длинных.");
}
```

Этот код делает практически то же, что и раньше, однако, если фамилия (surname) не длиннее 12 символов, он выводит другое, альтернативное сообщение.

Конструкция if... else похожа на конструкцию if, однако у нее целых два тела, между которыми расположено ключевое слово else. Первое тело будет выполнено, если условие дает true, иначе выполняется код второго тела.

**Цепочка из конструкций if... else**

Зачастую нужно проверить несколько условий и сделать что-то, если одно из них дает true. Пример: вы пришли в китайский ресторан и выбираете, что бы такое съесть. Больше всего вы любите курицу с лимоном (lemon chicken), и, если она есть в меню, вы ее закажете. Если же ее нет, вы закажете говядину в соусе из черных бобов (beef with black bean). Однако если и это блюдо отсутствует, вы остановитесь на свинине в кисло-сладком соусе (sweet and sour pork). Наконец, в маловероятном случае, когда нет ни одного из этих блюд, вы закажете рис с яйцом, поскольку знаете, что его подают во всех китайских ресторанах.

```
var lemonChicken = false;
var beefWithBlackBean = true;
var sweetAndSourPork = true;
if(lemonChicken)
{
console.log("Отлично! Я буду курицу с лимоном!")
}
else if(beefWithBlackBean)
{
console.log("Заказываю говядину. ");
}
else if(sweetAndSourPork)
{
console.log("Ладно, закажу свинину.");
}
else
{
console.log("Что ж, остается рис с яйцом.");
}
```

Чтобы создать цепочку if... else, начните с обычного оператора if и после закрывающей фигурной скобки его тела введите ключевые слова else if, а следом — еще одно условие и еще одно тело. После можно добавить еще else if, и так до тех пор, пока у вас не закончатся условия (которых может быть сколько угодно). Завершающая секция else будет выполнена, если ни одно из условий не дает true.

Имея цепочку if... else с завершающей секцией else, можно не сомневаться, что одно (и только одно) из тел будет выполнено. Как только выяснится, что одно из условий дает true, будет запущен код из соответствующего тела, а последующие условия проверяться уже не будут. Если запустить код из предыдущего примера, мы увидим в консоли «Заказываю говядину», поскольку beef With Black Bean — первое из усло-вий в цепочке if... else, которое равно true. Если же ни одно из условий не даст true, будет выполнено тело else.

Также обратите внимание: указывать завершающее else необязательно. Однако если вы этого не сделаете, то в случае, когда ни одно из условий не дает true, ничего из цепочки if... else выполнено не будет.

```
var lemonChicken = false;
var beefWithBlackBean = false;
var sweetAndSourPork = false;
if(lemonChicken)
{
console.log("Отлично! Я буду курицу с лимоном!")
}
else if(beefWithBlackBean)
{
```

```
console.log("Заказываю говядину. ");
}
else if(sweetAndSourPork)
{
console.log("Ладно, закажу свинину.");
}
```

В этом примере мы не стали указывать завершающую секцию else. Поскольку ни одного из ваших любимых блюд нет, в консоли не появится никаких сообщений (и, по всей видимости, вы останетесь без обеда, но ведь это и хорошо потому, что чревоугодие — это не есть хорошо).