Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«ПЕРМСКИЙ ГОСУДАРСТВЕННЫЙ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»

Н. Н. Василюк

ЯЗЫКИ ПРОГРАММИРОВАНИЯ ОСНОВЫ WEB-ПРОГРАММИРОВАНИЯ

Допущено методическим советом
Пермского государственного национального
исследовательского университета в качестве
учебного пособия для студентов, обучающихся
по направлению подготовки бакалавров
«Языки программирования»

Пермь 2019

УДК 004.43
ББК 32.973.4
В194

Василюк Н. Н.

В194 Языки программирования. Основы web-программирования: [Электронный ресурс]: учеб. пособие / Н. Н. Василюк; Перм. гос. нац. исслед. ун-т. Пермь, 2019. – 103 с. – Электрон. дан. – Пермь, 2019. – 2,57 Мб; 126 с. – Режим доступа: http://www.psu.ru/files/docs/science/books/uchebnie-posobiya/vasilyuk-yazyki-program-osnovy-web-program.pdf. – Загл. с экрана.

ISBN 978-5-7944-3279-4

Учебно-методическое пособие содержит теоретические сведения, примеры выполнения заданий и задания для самостоятельной работы студентов по теме «Web-программирование». В результате изучения изложенного в пособии материала студент должен получить представление об основных возможностях языка создания сценариев JavaScript, овладеть навыками их создания, познакомиться с основами программирования на языке PHP.

Предназначено для студентов направлений и специальностей, не специализирующихся на изучении информатики и информационных технологий.

УДК 004.43
ББК 32.973.4

Печатается по решению ученого совета механико-математического факультета Пермского государственного национального исследовательского университета

Рецензенты: кафедра информатики и ВТ ПГГПУ (рецензент – зав. кафедрой, канд. пед. наук, доцент ***А. П. Шестаков***);
доцент кафедры информационных технологий в бизнесе НИУ ВШЭ – Пермь, канд. пед. наук ***В. О. Кушев***

ISBN 978-5-7944-3279-4

СОДЕРЖАНИЕ

Введение

Веб-программирование – раздел программирования, ориентированный на разработку веб-приложений (программ, обеспечивающих функционирование динамических сайтов Всемирной паутины).

Языки веб-программирования – это языки, которые в основном предназначены для работы с веб-технологиями.

В результате изучения изложенного в пособии материала студент должен получить представление об основных возможностях языка создания сценариев JavaScript, овладеть навыками создания простейших сценариев, познакомиться с основами программирования на языке PHP.

Для освоения пособия студенту необходимо: иметь представление об основах создания Web-страниц и знать язык разметки гипертекста HTML.

Цель учебного пособия – познакомить студентов с основами программирования Web-страниц.

Методические указания

Примерное содержание практических занятий (16 ч.)

№ п/п	Тема	Количество часов
1	Знакомство с языком программирования JavaScript, работа с упражнениями	2
2	Работа с датами и массивами в языке JavaScript	2
3	Создание анкеты и скрипта на JavaScript, проверяющего правильность ее заполнения	4
4	Знакомство с языком программирования PHP, работа с упражнениями	2
5	Работа с числовыми и ассоциативными массивами в языке PHP	2
6	Работа с формами в языке PHP	4

Контрольные мероприятия

Работа студентов по изучению курса оценивается по результатам выполнения итогового задания. В качестве такого задания можно предложить по одной задаче из раздела «Задания для самостоятельной работы».

Глава 1. Язык разработки сценариев Java Script

1.1. Общие сведения о событиях и функциях обработки событий

Интерактивность Web-страниц состоит в том, что в ответ на какое-либо действие пользователя происходит изменение внешнего вида страницы, отображается дополнительная информация или выполняются другие процессы, облегчающие пользователю работу с документом. Действия пользователей, в ответ на которые происходит изменение страницы, называются ***событиями***. Чтобы назначить событию вызов сценария на выполнение, используются атрибуты событий элементов Web-страницы.

В таблице 1.1 представлены некоторые атрибуты событий, которым можно назначать выполнение сценариев.

Таблица 1.1

События, поддерживаемые элементами Web-страницы

Событие	Описание	Дескрипторы, поддерживающие событие
ONCLICK	Щелчок мышью на элементе	Большинство дескрипторов
ONDBCLICK	Двойной щелчок мышью на элементе	Большинство дескрипторов
ONMOUSEOVER	Пользователь наводит указатель мыши на элемент	Большинство дескрипторов
ONMOUSEOUT	Пользователь убирает указатель мыши с элемента	Большинство дескрипторов
ONFOCUS	Пользователь выбирает элемент с помощью клавиши <Tab>	<A>, <AREA>, <LABLE>, <INPUT>, <SELECT>, <BUTTON>, <TEXTAREA>
ONBLUR	Элемент теряет выделение при следующем нажатии клавиши <Tab>	<A>, <AREA>, <LABLE>, <INPUT>, <SELECT>, <BUTTON>, <TEXTAREA>
ONSELECT	Пользователь выделяет текст элемента	<INPUT>, <TEXTAREA>
ONCHANGE	Пользователь изменяет текст элемента	<INPUT>, <SELECT>, <TEXTAREA>
ONSUBMIT	Пользователь щелкнул в форме на кнопке «Подача запроса»	<FORM>
ONRESET	Пользователь щелкнул в форме на кнопке «Сбросить»	<FORM>
ONLOAD	Web-страница загружается в окно обозревателя или в рамку	<BODY>, <FRAMESET>, <FRAME>
ONUNLOAD	Web-страница замещается другой страницей в окне браузера	<BODY>, <FRAMESET>, <FRAME>

Именование элементов Web-страницы

Для любого элемента Web-страницы можно задать уникальное имя, присвоив его атрибуту ID и/или NAME.

Например, в следующем примере присваиваем элементу основного раздела страницы имя body:

```
<BODY  ID='body'   NAME='body'>
```

Смысл именования элементов состоит в том, что разработчик получает возможность в сценариях обращаться к элементам Web-страницы по имени и динамически изменять их атрибуты. Например, после того как мы присвоили разделу <BODY> имя, в коде сценария можно установить или изменить цвет фона страницы, используя следующую команду:

```
window.body.bgColor='red'
```

Эту команду можно присвоить атрибуту события, например событию ONCLICK кнопки формы.

Создание пользовательских функций

В сценариях часто бывает необходимо выполнить не одну, а несколько команд. В таком случае удобно создавать и применять ***пользовательские функции.***

Сценарии выполняются во время загрузки Web-страницы в той очередности, в которой они находятся в коде HTML. Но если команды и стандартные функции JavaScript в сценариях выполняются автоматически, то последовательности команд функций сохраняются в оперативной памяти компьютера под именем функции и выполняются только в том случае, если функция была вызвана на выполнение.

Пользовательские функции создаются с помощью ключевого слова **function** следующим образом:

```
<SCRIPT LANGUAGE='JavaScript'>
function ИмяФункции{список аргументов)  {
код функции
};
др.  функции …
</SCRIPT>
```

Определения функций всегда отделяются от остального кода Web-страницы с помощью пары дескрипторов сценария <SCRIPT>... </ SCRIPT >.

За ключевым словом **function** следует имя функции. Имя функции должно быть уникальным для данной Web-страницы. Если в текущем или следую-

щем сценарии на Web-странице будет определена другая функция с таким же именем, то прежняя функция будет замещена новой.

Имя функции завершается парой скобок, между которыми находится список аргументов функции. Список аргументов может быть пустым «()», или содержать имена аргументов, используемых в данной функции: ***ИмяФункции(a, b, c, ...)***. В соответствии с синтаксисом JavaScript код функции заключается в фигурные скобки {...}, а каждая строка кода должна завершаться символом точки с запятой «;».

После того как функция была определена, ее можно вызвать на выполнение в коде сценария следующим образом:

ИмяФункции(список аргументов);

Число аргументов в вызове функции должно соответствовать числу аргументов в определении функции.

Чаще всего функции используют для обработки событий, выполняемых над документом HTML или его элементами. Функция, назначенная атрибуту события, называется ***функцией обработки события.*** При использовании гиперссылок функции могут назначаться как атрибутам событий, так и непосредственно HREF. Но при этом используется особый синтаксис назначения функции с помощью ключевого слова **javascript**:

```
<A href='javascript:def_show()'>Текст гиперссылки</A>
```

В этом случае текст будет выглядеть как обычная гиперссылка, но функционировать как кнопка формы.

Если вы хотите, чтобы во время щелчка на гиперссылке одновременно с открытием новой страницы выполнялась какая-нибудь дополнительная функция, назначьте эту функцию событию ONCLICK данной гиперссылки.

Для возвращения полученного значения в функции используется команда **return**. Например, ниже показана функция для вычисления суммы двух чисел:

```
function sum(a, b) {
return (a + b); }
```

Динамическое редактирование блочных элементов Web-страницы

С помощью функций можно также изменять или возвращать содержимое блочных элементов Web-страницы. Для этого используются атрибуты **innerText** и **innerHTML**. Чтобы понять смысл этих атрибутов, рассмотрим

два следующих примера абзацев текста, которые на Web-странице будут выглядеть совершенно одинаково:

```
<P>Текст абзаца</P>
<P innerText='Текст абзаца'></P>
```

Таким образом, атрибут содержит весь текст блочного элемента, в данном случае абзаца. Атрибут innerHTML отличается тем, что содержит не только текст, но и другие дескрипторы HTML, такие, как дескрипторы форматирования и вложенные блочные элементы. Рассмотрим примеры динамического изменения текста Web-страницы.

1.2. Ввод и редактирование кода HTML с помощью сценариев

В предыдущих разделах указывалось, что элементы Web-страницы, созданные с использованием дескрипторов HTML, могут быть изменены динамически уже после открытия Web-страницы с помощью сценариев. В этом разделе речь пойдет о динамическом создании кода HTML посредством сценариев, т.е. о создании новых Web-страниц по требованию пользователей.

Функция write

Функция write языка JavaScript используется для вывода динамического текста и дескрипторов HTML в окно обозревателя. Синтаксис пользования этой функции следующий:

```
document.write ("текст")
```

Прежде всего нужно указать документ, в котором следует ввести новый текст. Ключевое слово document соответствует текущему окну обозревателя. Если текст вводится в определенную рамку или в другое окно обозревателя, то перед словом document необходимо указать имя рамки или окна:

```
main.document.write("текст")
NewWin.document.write("текст")
```

Аргументом функции может быть строка текста, число или вызов функции JavaScript (стандартной или пользовательской), возвращающей текстовое или числовое значение. Каждая последующая функция write вводит текст, вслед за текстом, добавленным предыдущей функцией. Чтобы создать новый абзац, используйте в строке текста дескрипторы HTML:

```
document.write("<P>Первый абзац</P>")
document.write("<P>Второй абзац</P>")
```

или одну команду:

document.write("<P>Первый абзац</P><P>Второй абзац</P>")

В строке функции write можно использовать любые дескрипторы как для форматирования текста, так и для создания таблиц, форм, списков и других элементов Web-страницы.

Функции write можно передать несколько аргументов, разделив их запятыми. Функция автоматически объединит строки текста, переданные в аргументах, и выведет суммарную строку в окне обозревателя. Это свойство удобно использовать для включения в текст результатов выполнения пользовательских функций, как в следующем примере:

WinCulc.document.write("<P>Сумма налога за автомобиль составит", culc__tax(car__type. value, year .value) ," рублей</P>")

Функция write выводит результат одним абзацем в новое окно вычислений с сопровождающим текстом обозревателя с именем *WinCulc*. Данная функция выводит строку с указанием суммы налога за автомобиль. Предположим, что на Web-странице есть форма с двумя полями *Тип автомобиля* и *Год выпуска*, которым присвоены имена *car_type* и *year* соответственно. Вам не составит труда создать такую форму самостоятельно. В Web-странице также есть сценарий с определением функции *culc_tax*, которая возвращает значение, рассчитанное по данным из полей формы.

Очень важно правильно выбрать место функции write в сценарии на Web-странице, чтобы получить требуемый результат. Если функция document. write используется в сценарии, то строки текста будут добавляться за элементами Web-страницы, созданными ранее в коде HTML. Функцию write также можно применять внутри пользовательских функций обработки событий. Обычно в этом случае функция write применяется для заполнения текстом определенных рамок или окон обозревателя.

Создание объектов элементов Web-страницы

Для создания объектов Web-страницы используется функция **createElement**. Синтаксис функции: в аргументе указывается дескриптор HTML соответствующего объекта в кавычках, но без угловых скобок <...>. Например:

•document. createElement (' P ') – создает новый абзац, аналогичен <P></P>;

•document. createElement ('TABLE') – создает новую таблицу, аналогичен <TABLE></TABLE>;

•document. createElement ('TD') – создает новую ячейку таблицы, аналогичен <TD></TD> и т.д.

Команда **createElement** создает новый объект в оперативной памяти компьютера. Теперь этот объект можно добавить в определенное место на Web-странице. Для примера в листинге 1.1. создается новый абзац текста.

Листинг 1.1

Динамическое создание и добавление объектов Web-страницы

```
<BODY>
<DIV ID='tag' NAME='tag'></DIV>
<SCRIPT>
par = document.createElement(" P") ;
par.innerText = "Текст абзаца";
tag.appendchild(par);
</SCRIPT>
</BODY>
```

При загрузке Web-страницы с помощью дескриптора <DIV ID =' tag' NAME=' tag' > создается пустой именованный раздел, после чего выполняется код сценария. В сценарии создается новый объект абзаца под именем **par**, которому присваивается текст: par. innerText = " Текст абзаца".

В следующей строке программы абзац добавляется в раздел **tag** с помощью команды tag.appendChild(par). В данном случае команда appendChild добавляет *дочерний* объект в *родительский* блочный объект. Эта команда применима ко всем объектам, однако необходимо учитывать логику отношений дочерних и родительских объектов. Объект абзаца не может быть дочерним по отношению к объекту другого абзаца, а объект ячейки таблицы может быть дочерним по отношению к объекту строки таблицы, но не самой таблицы.

1.3. Сообщения для посетителей Web-страниц

JavaScript предоставляет различные средства для организации диалога между создателем и посетителями веб-страницы. Наиболее эффективны в этом плане диалоговые окна, с помощью которых можно как сообщить информацию, так и попросить подтверждения действия или задать вопрос посетителю. Кроме

того, сообщения для пользователей можно показывать в строке состояния и строке заголовка окна обозревателя.

Использование диалоговых окон

В JavaScript используются три встроенных диалоговых окна. Заголовки и набор кнопок в этих окнах постоянны. Текст сообщения задается аргументом функции открытия диалогового окна.

- **alert (**'**сообщение**') – открывает диалоговое окно с текстом сообщения и единственной кнопкой **ОК** (рис. 1.1). В качестве сообщения можно использовать обычные строки текста, заключенные в кавычки, цифры, переменные и вызовы функций, возвращающие текстовые или цифровые значения. Это диалоговое окно применяется для показа предупреждений или информационных сообщений, не требующих от пользователя принятия каких-либо решений.
- Функцию alert удобно использовать для отладки сценариев. Поместите функцию alert в любой строке кода сценария, чтобы определить сбойную строку или отобразить текущие значения переменных.
- **confirm(**'**сообщение**') – открывает диалоговое окно с текстом сообщения и двумя кнопками – **ОК** и **Cancel** (рис. 1.2). Обычно в сообщении пользователю предлагается подтвердить выполнение операции, щелкнув на кнопке ОК, или отказаться от выполнения щелчком на кнопке **Cancel.** В зависимости от выбора кнопки функция confirm возвращает значение true (**ОК**) или false **(Cancel).** Возвращенное значение затем анализируется в конструкции с оператором if.
- **prompt ('сообщение',' текст по умолчанию**') – открывает окно для ввода данных пользователем. Кроме кнопок **ОК** и **Cancel** данное окно содержит текстовое поле ввода (рис. 1.3). В текстовом поле по умолчанию отображается текст, заданный во втором аргументе функции prompt. Чтобы оставить текстовое поле пустым, введите во втором аргументе пустую строку – "". Если второй аргумент пропустить, то в текстовом поле отобразится текст ***undefined,*** как показано на рис. 1.10. После щелчка на кнопке **ОК** функция prompt возвращает значение текстового поля, а после щелчка на кнопке **Cancel** – значение false.

Пример использования диалоговых окон показан в листинге 1.2.

Листинг 1.2

Применение стандартных диалоговых окон в коде сценария

```
<HTML>
<HEAD>
<TITLE>Диалоговые окна</TITLE>
<SCRIPT>
function do_payment(cart) {
alert(cart);
}
function finish() {
alert("До свидания, вы покидаете систему электронных платежей!");
}
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
alert("Вы вошли в систему электронных платежей!");
if(confirm("Хотите ли вы выполнить платеж?")) {
cart = prompt("Введите номер кредитной карточки."); if (confirm(
"Вы ввели номер карточки " + cart + ", пожалуйста подтвердите."))
do_payment(cart);
else
finish();
}
else
finish();
</SCRIPT>
</BODY>
</HTML>
```

Основной раздел содержит сценарий, который выполняется автоматически во время загрузки Web-страницы. В первой строке сценария функция alert открывает окно сообщения, показанное на рис. 1.1.

Далее выполняется конструкция if (confirm("Хотите ли вы выполнить платеж?")). Функция confirm открывает диалоговое окно, показанное на рис. 1.1.

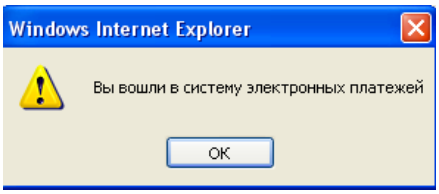

Рис. 1.1. *Окно сообщения*

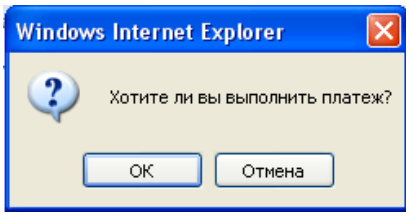

***Рис. 1.2**. Окно подтверждения*

После щелчка на кнопке **ОК** выполняются команды, следующие за конструкцией с оператором if, а если щелкнуть на кнопке **Cancel** – выполняются команды, следующие за оператором else. Таким образом, если вы подтвердите желание выполнить платеж, следующая команда prompt("Введите номер кредитной карточки.") откроет диалоговое окно, показанное на рис. 1.3.

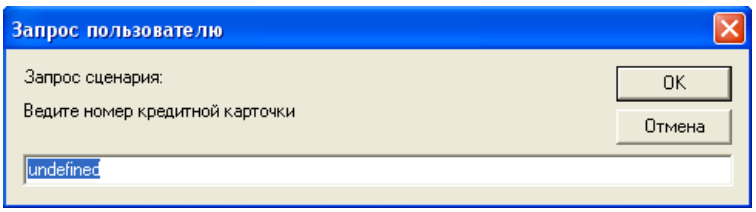

***Рис. 1.3.** Окно запроса сценария*

Введите номер карточки и щелкните на кнопке **ОК**. Введенное число будет сохранено в переменной *cart*.

Далее в сценарии следует еще одна конструкция с оператором if и функцией confirm. Обратите также внимание на то, что в функции confirm текст сообщения формируется из строк и значения переменной *cart*:

```
confirm("Вы ввели номер карточки " + cart + ", пожалуйста, подтвердите.")
```

Результатом выполнения сценария будет либо вызов функции *do_payment*, в которую передается номер карточки (в нашем примере эта функция просто покажет номер в окне сообщения), либо вызов функции finish, которая сообщает о завершении выполнения программы.

Сообщения в строке состояния

Для управления текстом строки состояния используются два метода:

- defaultStatus = text – устанавливает текст строки состояния по умолчанию. Обычно эта установка производится во время загрузки Web-страницы в сценарии основного раздела.
- status = text – используется в функциях обработки событий для установки контекстно-зависимых сообщений.

Строка состояния принадлежит объекту window, поэтому данные методы вызываются либо для объекта окна обозревателя по его имени, либо для теку-

щего окна с ключевым словом self или window. Например, ниже показан синтаксис обработки события ONFOCUS:

ONFOCUS = "self.status = 'Введите свой адрес электронной почты' "

ONFOCUS = "window.status = 'Введите свой адрес электронной почты' "

ONFOCUS = "MyWindow.status = 'Введите свой адрес электронной почты' "

В первых двух случаях устанавливается текст строки состояния текущего окна, а в третьем случае – строка состояния окна с именем MyWindow.

В случае использования метода status для обработки событий ONMOUSEOVER И ONMOUSEOUT функция должна завершаться командой **return true**:

ONMOUSEOVER = "self.status = 'Моя электронная почта'; return true;"

1.4. Динамическое форматирование элементов Web-страницы

Установка атрибутов в коде HTML

Чтобы получить доступ к атрибутам элемента Web-страницы из кода сценария, необходимо присвоить этому элементу уникальное имя, установив его в дескрипторе элемента для атрибутов ID и NAME. Если уникальное имя задано, то значение атрибута можно изменить динамически в коде сценария следующим образом:

имя_элемента.атрибут = значение

Например, скроем границы таблицы с именем tb:

tb.border = 0

Гораздо больше возможностей для форматирования элементов Web-страницы предоставляет атрибут STYLE. Вы знаете, что значением атрибута STYLE является текстовая строка со списком параметров форматирования и их значений, например:

<P ID=par1 NAME=par1 STYLE='color: red; font-style: italic'>

В коде сценария JavaScript стиль объекта можно изменить двумя способами: присвоить атрибуту STYLE строку текста с параметрами так же, как в коде HTML, или обратиться непосредственно к параметру, как в следующем примере:

par1.style.color = "FFFFOO";

Изменение цвета фона и гиперссылок Web-страницы

В коде HTML можно именовать все объекты Web-страницы, создаваемые с помощью соответствующих дескрипторов. Но именование никогда не применяется к обязательным дескрипторам:

- <HTML> – соответствует объекту window;
- дескрипторы заголовка Web-страницы и <BODY> – соответствуют объекту document.

Например, для доступа к тексту заголовка использовался следующий код (см. раздел «Динамическое изменение заголовков окна обозревателя» этой главы):

document.title = *текст;*

Доступ к атрибутам, устанавливаемым в дескрипторе <BODY>, также можно получить посредством объекта document:

- document. bgColor – цвет фона;
- document. fgColor – цвет текста страницы по умолчанию;
- document. 1inkColor – цвет неиспользованной гиперссылки;
- document. alinkColor – цвет активной гиперссылки в тот момент, когда на нее наведен указатель мыши;
- document. vlinkColor – цвет использованной гиперссылки.

В случае изменения настроек цвета в одной из рамок или в другом окне обозревателя перед словом document нужно указать имя рамки или окна:

win2. document. bgColor = blue-main, document.1inkColor = white;

1.5. Работа с данными в сценариях JavaScript

Создание переменных

JavaScript представляет собой *упрощенную версию языка программирования*. Объявление переменных происходит автоматически при присвоении им первого значения. Более того, JavaScript позволяет динамически изменять тип переменной, просто присвоив ей значение иного типа.

Для объявления переменных в JavaScript используется ключевое слово **var**, за ним следует имя переменной. Тип переменной устанавливается автоматически в ходе присвоения переменной первого значения. Оператором присваивания служит знак равенства (=). Значение, находящееся справа от знака равенства, присваивается переменной слева. Справа от знака равенства может находиться:

- **константное значение** – число или текст (текстовое значение должно быть заключено в кавычки;
- **вызов функции** – в примере выше использовались функция prompt, открывающее диалоговое окно и окно, возвращающее текст, введенный пользователем, а также функция Date, возвращающая объект текущей даты;
- **другая переменная** – имя переменной, которая была объявлена ранее, в том числе атрибуты элементов Web-страницы, например, значение текстового поля, как в примере выше.

Хотя объявления можно пропустить, код сценария станет понятнее, если в самом начале будет дан список используемых переменных:

```
var name = prompt{"Введите ваше имя."); // имя пользователя
var amount = currency.value; // сумма денег, взятая из поля "Сумма"
var date = new Dated; // объект текущей даты
var rate = 5.05; // курс
```

Примечание. После двух слэшей (//) в сценарии JavaScript записываются комментарии, не влияющие на ход работы сценария.

Имя переменной

В JavaScript следует придерживаться определенных правил при именовании переменных:

- в имени можно использовать латинские буквы, цифры и подстрочный символ «_»;
- имя переменной должно начинаться с буквы или с подстрочного символа, но не с цифры;
- в имени переменной нельзя использовать знаки пунктуации, специальные символы, а также пробелы;
- имя переменной чувствительно к регистру букв (sum и Sum будут рассматриваться как две разные переменные);
- длина имени переменной не ограничена.

В сценарии у всех переменных с перекрывающимися областями видимости должны быть уникальные имена. Если в сценарии будет объявлена переменная с таким же именем, как у ранее созданной переменной, новая переменная заменит собой предыдущую.

Типы данных

В JavaScript различают следующие типы данных:

- целое число;
- число с плавающей запятой;
- строка текста;
- логическая переменная;
- объект

Целые числа

Переменные этого типа могут содержать целочисленные значения, как положительные, так и отрицательные. В листинге 1.3 показан пример объявления и использования целочисленных переменных.

Листинг 1.3

Использование целых чисел

```
<HTML>
<HEAD>
<TITLE>Целые числа</TITLE>
</HEAD>
<BODY>
<SCRIPT>
// Объявления переменных
var iVal =14;      // десятеричное число
var hVal = OxE;  // шестнадцатеричное число
var oVal = 016;  // восьмеричное число
// Вывод переменных в окне обозревателя
document.write("Это десятеричное число: " + iVal + "<BR>");
document.write("Это шестнадцатеричное число: "
+ hVal + "<BR>"); document.write("Это восьмеричное число: " + oVal
+ "<BR>"); document.write("Это сумма чисел: " + (oVal + hVal) +
"<BR>");
</SCRIPT>
</BODY>
</HTML>
```

Обратите внимание: функция write выводит все значения как десятеричные числа. Целые числа можно суммировать и использовать в других вычислениях независимо от того, в какой нумерической системе они были введены.

Числа с плавающей запятой

Числовые значения этого типа могут представлять целую и дробную часть числа, отделенные точкой (в JavaScript число 3,2 следует вводить как 3.2).

Числа с плавающей запятой в JavaScript могут быть представлены в двух форматах: обычном и логарифмическом. Логарифмический формат имеет следующий синтаксис: **#.##E??**

Например, число 5 140 000.0 можно представить как 5.14E6, а число 0.0006023 – как 6.023E-4.

JavaScript автоматически выводит в окне обозревателя числа с плавающей запятой в логарифмическом формате, если число превышает значение 1.0E20 или меньше числа 1.0E-7.

Строки текста

Строки текста могут содержать последовательности любых символов, в том числе и цифр. Но в этом случае цифры будут рассматриваться программой как текст, а не как числовые значения.

Чтобы присвоить текст переменной, строку следует заключить в кавычки. Можно использовать как одинарные ('), так и двойные (") кавычки, но они должны быть одинаковыми в начале и в конце строки. Если строка заключена в двойные кавычки, то в тексте можно использовать одинарные. Чтобы ввести в текст двойные кавычки, используйте символ обратного слеша: \". Этот символ указывает программе, что за ним идет обычный текст, а не управляющий символ. Тогда добавление в текст одного символа обратного слеша будет выглядеть так: \\.

Если текст выводится в окно обозревателя с помощью команды document .write, то в нем для форматирования можно использовать дескрипторы HTML. Ниже приведен фрагмент программного кода присвоения текста переменной и вывода значения на экран. Результат выполнения программы показан на рис. 1.4.

```
sVal = "В тексте мы можем использовать<I>\"кавычки\"</I>, заключать слова в <B>\<угловые\></B> или \\косые\\ скобки.";
document.write(sVal);
```

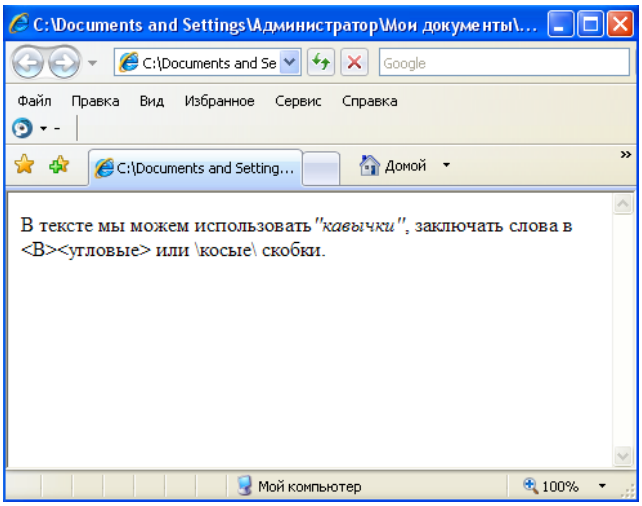

Рис. 1.4. *Вывод текстовой переменной в окно обозревателя*

Логические переменные

Логические переменные могут принимать только значения true (истинно) и false (ложно). Впрочем, этим переменным также можно присваивать целочисленные значения, при этом 0 будет соответствовать false, а любое другое число – true. Обычно логические переменные используются в логических конструкциях управления выполнением программы с использованием операторов if и else.

Область видимости переменных

Под *областью видимости* переменных подразумевают часть программного кода, в которой переменная может использоваться. При создании переменной выделяются ячейки памяти для хранения ее данных. Как только необходимость в переменной исчезает, память освобождается и становится доступной для записи других переменных. В JavaScript нет средств, подобных тем, что имеются в профессиональных языках программирования, которые позволяли бы управлять памятью компьютера, поэтому выделение ячеек памяти и их очистка происходят автоматически. Разработчик сценариев должен точно знать, в какой части программного кода переменная доступна, а в какой – нет.

В зависимости от области видимости различают два вида переменных.

- **Глобальные переменные** – это переменные, созданные непосредственно в коде сценария в блоке <SCRIPT>...</SCRIPT> где-либо в коде Web-страницы. Во время загрузки Web-страницы эти переменные доступны для использования в функциях и сценариях, код которых находится ниже в коде HTML Web-страницы. После окончания загрузки Web-страницы эти перемен-

ные доступны для всех функций текущего документа, а также для сценариев, запускаемых в других окнах обозревателя, если есть возможность обратиться к текущему окну по имени: *ИмяОкна.имя_переменной*.

Поскольку сценарии выполняются по мере загрузки Web-страницы, глобальные переменные следует объявлять раньше, чем они будут использованы в данном сценарии, другом сценарии или в функции. Например, предположим, что есть функция funk(), в которой используется глобальная переменная gval.

В табл. 1.2 показано правильное и ошибочное использование переменной.

Таблица 1.2

Примеры использования глобальной переменной

Правильное использование	Ошибочное использование

var gVal = 10;
… код сценария
funk();
	funk();
	... код сценария
	var gVal = 10;

- **Локальные переменные** – это переменные, которые объявлены в коде функции. Использование таких переменных ограничено кодом функции ниже того места, где они были объявлены. После завершения функции все ее локальные переменные автоматически удаляются из памяти компьютера, поэтому не могут использоваться другими функциями или командами сценария, даже в том же сценарии, где функция была определена или вызвана на выполнение.

Объявляйте локальные переменные в функциях с помощью ключевого слова var, чтобы избежать *конфликта имен* с одноименными глобальными переменными. Если переменная в функции создается путем простого присвоения значения и в данной странице есть глобальная переменная с таким же именем, то в действительности будет происходить не создание новой переменной, а присвоение глобальной переменной нового значения.

В листинге 1.4 показан пример правильного и ошибочного использования локальных переменных.

Листинг 1.4

Использование локальных и глобальных переменных

```
<HTML> <HEAD>
<TITLE>Использование переменных</ TITLE >
<SCRIPT>
//В функциях используются глобальная и локальная переменные
function right() {
var  inText
inText = prompt("Введите ваш текст:");
if (inText)
alert(gText + inText);
}

function wrong() {
alert(gText + inText);
}

</SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
// Глобальная переменная
var gText = "Был введен следующий текст: ";
</SCRIPT>
<FORM>
<INPUT TYPE='button' VALUE="Правильное использование пере-
менной" ONCLICK='right()'><BR>
<INPUT TYPE='button' VALUE="Ошибочное использование
переменной" ONCLICK='wrong()'>
</FORM>
<BODY>
</HTML>
```

В сценарии основного раздела Web-страницы определена глобальная переменная gVal, которой присвоена строка "Был введен следующий текст:". Затем создаются две кнопки Правильное использование переменной и Ошибочное использование переменной, вызывающие на выполнение функции right и wrong соответственно. Первая функция сначала открывает диалоговое окно с предложением ввести текст, сохраняет этот текст в локальной переменной inText, а затем показывает в окне сообщения строку, состоящую из объединенных значений глобальной и локальной переменных (рис. 1.5). Обратите внимание: функция работает, несмотря на то, что в коде HTML объявление глобаль-

ной переменной находится ниже определения функции, поскольку вызов функции происходит уже после того, как Web-страница будет загружена в окно обозревателя.

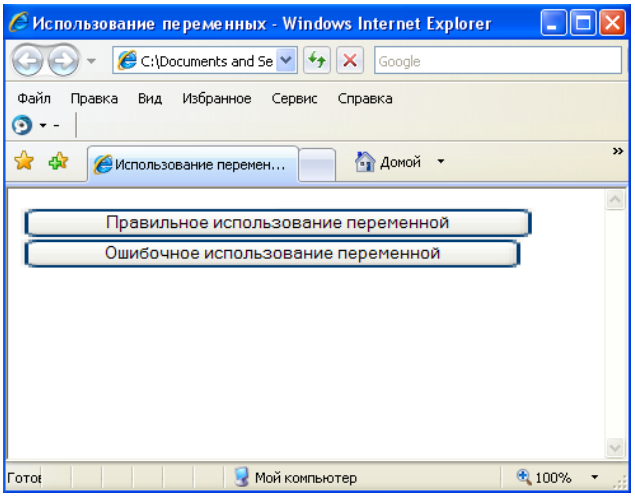

***Рис. 1.5**. Правильное и ошибочное использование локальных переменных*

Если щелкнуть на другой кнопке, то в строке состояния окна обозревателя появится сообщение: *Ошибка на странице*. Ошибка возникла вследствие того, что функция обращается к переменной inText, которая определена в другой функции и не существует в данной.

Динамическое определение типа переменной

Возможность смены типа переменной в ходе выполнения сценария JavaScript создает ситуацию, которая невозможна во многих других языках программирования. Иногда нельзя предугадать, каким будет тип переменной перед ее использованием. Например, можно предложить пользователю ввести в поле текст или число, после чего выполнить ветвление программы в зависимости от типа введенных данных. Для определения типа переменной в коде сценария используется команда **typeof** *имя_переменной*. Команда возвращает текстовое значение, соответствующее типу переменной:

- **number** – целое число или число с плавающей запятой;
- **string** – строка текста;
- **boolen** – логическое значение;
- **object** – объект;
- **underfined** – переменная не определена (не была создана или программа вышла за область видимости переменной).

Обычно проверку типа переменной выполняют в конструкции с оператором if, например:

```
if (typeof val == 'string')
{команды обработки строкового значения}
else
{команды обработки значения другого типа}
```

Преобразования переменных

Чаще всего конфликты типов данных возникают при использовании строк вместо чисел и наоборот. Иногда бывает необходимо число представить как строку или, скажем, вырезать из строки часть информации и преобразовать ее в число. Для выполнения этих операций в JavaScript предусмотрен ряд функций.

Чтобы преобразовать числа в строки текста, используется функция toString (). Вот пример использования этой функции:

```
var iVal = 5;
var sVal = i Val .toString ();
```

В результате получим переменную sVal, значением которой будет строка ' 5 '.

Строковую переменную из числа можно получить, прибавив число к пустой строке:

```
var sVal = '' + iVal;
```

Результат будет таким же, как и в предыдущем примере.

Для выполнения обратного действия используются функции parseInt() и parseFloat(), например:

```
var sVal = '5';
var iVal = parseInt(sVal);
var fVal = parseFloat(sVal);
```

Первая функция возвращает целое число, а вторая – число с плавающей запятой.

1.6. Объекты и массивы данных

Массивы содержат коллекции взаимосвязанных данных. В JavaScript, в отличие от многих других языков программирования, один массив может содержать данные разных типов. Можно создать массив значений, а также масси-

вы объектов других типов (в том числе и массивов – *многомерные массивы),* массивы методов, гиперссылок, файлов мультимедиа и т.д.

Объекты представляют собой более сложно организованные массивы данных, поскольку они кроме значений разных типов содержат также *методы* обработки этих значений.

Методами называются функции, являющиеся частью объекта. Вызов метода происходит следующим образом: *имя_объекта.метод().* Как видите, с методами вам уже приходилось иметь дело, например с методом write, который принадлежит объекту document.

Общим для объектов JavaScript является использование ключевого слова new при создании нового *экземпляра* объекта. Команда typeof, вызванная для экземпляра объекта, возвращает значение object.

Создание и использование массивов

Массив представляет собой особый тип переменной, содержащей не одно, а много значений, называемых *элементами* массива. Элементы массива хранятся в той последовательности, в которой они добавлялись в массив. К любому элементу можно обратиться по имени массива, указав порядковый номер элемента, называемый *индексом.* Синтаксис обращения следующий:

элемент = имя_массива[индекс] ;

Нумерация элементов массива начинается с нуля. Таким образом:

первый_элемент = имя_массива[0];

последний_элемент = имя_массива[размер_массива – 1];

Индекс может быть задан числом, целочисленной переменной или функцией, возвращающей целочисленное значение.

Установка в индексе числа, превышающего размер массива, приведет к ошибке выполнения сценария. Чаще всего ошибка происходит при обращении к последнему элементу массива, когда в качестве индекса используют значение размера массива, в то время как индекс последнего элемента на единицу меньше.

Новый массив создается с помощью ключевого слова new: array1 = new Array (#), где # – размер массива.

Размер массива указывает число элементов массива. Так, в строке array1 = new Array (10) создается массив из десяти элементов с array1 [0] по array1 [9].

При создании объектов в JavaScript используется следующий синтаксис: new *класс_объекта (аргументы).* Эту функцию называют *конструктором класса.* Обратите внимание на то, что имя конструктора класса чувствительно к регистру букв.

При создании массива можно задать его размер. Впрочем, размер массива очень просто изменить в ходе выполнения сценария. Если размер не указан, создается пустой массив.

Следующий этап состоит в заполнении элементов массива данными, как в следующем примере:

```
months = new Array(12);
months[0] = "январь";
months[1] = "февраль";
```

Мы заполнили два первых элемента массива названиями месяцев года. Остальные 10 элементов остались пустыми. Если сейчас с помощью функции write или alert мы выведем содержимое третьего элемента, months [2], то отобразится значение undefined.

Существует другой способ создания массива одновременно с присвоением значений всем его элементам:

```
months = new Array("январь", "февраль", "март", "апрель", "май", "июнь", "июль", "август", "сентябрь", "октябрь", "ноябрь");
```

Методы массивов

Массив, как любой другой стандартный объект JavaScript, содержит подборку полезных методов, которые предназначены для преобразования массива в строку и для изменения порядка следования элементов.

Изменение размера массива и порядка следования элементов

Увеличить размер массива очень просто. Присвойте значение элементу с индексом, превышающим текущий размер массива. Массив автоматически будет увеличен до элемента с максимальным индексом. Например, в следующем примере создается массив из 10 элементов, причем значение присвоено только последнему элементу.

```
Array1 = new Array();
Array1[9] = "значение последнего элемента";
```

Все остальные элементы массива остались неопределенными.

Значение размера массива хранится в переменной length, встроенной в объект массива. В любой момент мы можем определить текущий размер массива и использовать это значение, например вывести в окне сообщения: alert(month.length);

Обратите внимание на то, что length – это не метод, а имя переменной. Переменные, встроенные в объект, называются *переменными-членами* или *свойствами* объекта. В переменной length хранится информация о числе элементов массива. Максимальный индекс массива можно вычислить так:

max_index = *имя_массива*.length – 1;

Объект Array содержит ряд методов, которые позволяют изменять структуру массивов:

- **concat (массив2)** – объединяет два массива и возвращает новый массив, в котором элементы массива 2 следуют за элементами исходного;
- **slice (индекс1,индекс2)** – возвращает часть массива от элемента с индексом 1 до элемента с индексом 2, не включая его (чтобы скопировать массив до конца, просто пропустите второй аргумент);
- **splice (индекс1,индекс2)** – работает так же, как и предыдущий метод, но извлекаемый подмассив удаляется из исходного массива;
- **push (значение)** – добавляет новый элемент в конец массива;
- **pop** () – возвращает последний элемент и удаляет его из массива;
- **shift** () – возвращает первый элемент и удаляет его из массива;
- **unshift (значение)** – добавляет новый элемент в начало массива;
- **reverse** () – инвертирует порядок следования элементов в массиве;
- **sort** () – сортирует элементы массива в алфавитном порядке.

При сортировке текстовых элементов массива следует учесть, что на нее повлияет регистр букв.

Преобразование массива в строку и поиск элементов

Для преобразования массива в строку можно использовать два метода: toString () и join (). Оба метода по умолчанию возвращают список элементов, разделенных запятыми (рис. 2.3). Метод join () позволяет установить другой разделитель, указав его в аргументе функции, например months. join ('; ').

1.7. Стандартные массивы Web-страницы

Любая Web-страница уже содержит ряд стандартных встроенных массивов, в которых представлены коллекции ее элементов. Имена массивов и их иерархия показаны на рис. 1.6.

Элементы Web-страницы автоматически добавляются в соответствующие массивы в той последовательности, в которой они создаются в коде HTML. Все массивы элементов принадлежат объекту document (соответственно, к ним можно обратиться: document *.имя_массива),* но массив рамок frames можно создать непосредственно из документа window: имя_окна. frames [0].

Удобство использования массивов элементов состоит в том, что с их помощью можно установить любой элемент Web-страницы, даже если для него не было задано уникальное имя.

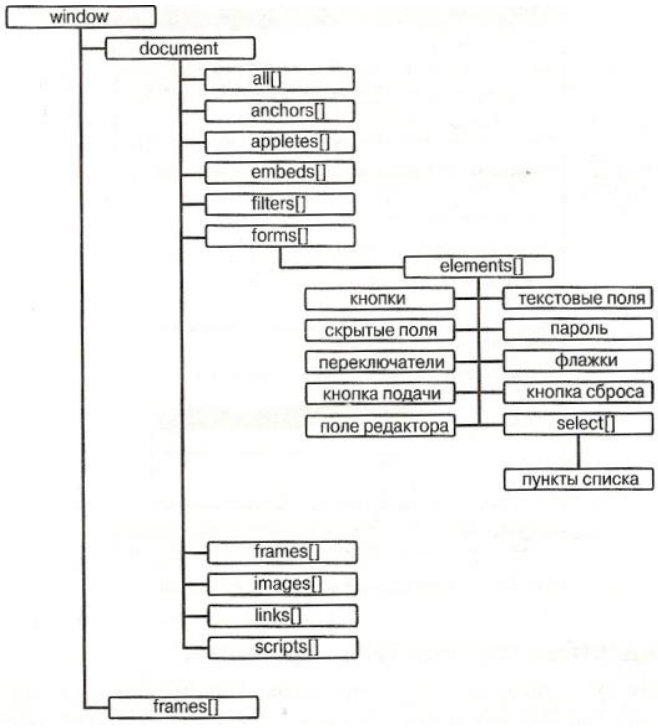

Рис. 1.6*. Стандартные массивы элементов Web-страницы*

Обращение к элементам Web-страницы по именам все же предпочтительнее, так как число и порядок элементов в массиве могут измениться в результате динамического изменения самой Web-страницы.

Наиболее часто используются следующие массивы элементов:

- **all** – все элементы Web-страницы;
- **frames** – рамки;
- **images** – рисунки;
- **links** – гиперссылки;

- **forms** – формы;
- **elements** – элементы формы.

1.8. Выражения и операции

Сценарии в Web-страницах часто используют для того, чтобы произвести анализ данных, введенных пользователем, и возвратить результат. Для выполнения операций с переменными используются операторы языка JavaScript.

Выделяют следующие группы операторов:

- арифметические;
- присваивания;
- сравнения;
- логические;
- строковые.

Арифметические операции

В сценариях JavaScript можно выполнять простые и довольно сложные вычисления с помощью стандартных операторов JavaScript, а также с помощью методов объекта Math. Арифметические операторы применяются для создания выражений, в которых помимо операторов используются переменные (целочисленные и с плавающей запятой) и собственно числовые значения, называемые *литералами.* Обычно выражение также содержит оператор присваивания, который присваивает результат вычисления, находящийся справа от него, переменной, находящейся слева. Пример выражения показан на рис. 1.7.

***Рис. 1.7**. Пример арифметического выражения в сценарии JavaScript*

Выполнение базовых математических операций

В JavaScript используются 5 арифметических операторов:

+ – суммирования;

– вычитания;

***** умножения;

/ – деления;

% – вычисление модуля.

Первые четыре оператора не требуют объяснений. Оператор вычисления модуля используется для извлечения остатка от деления.

Операторы инкремента

В сценариях JavaScript часто используются счетчики – переменные, значение которых изменяется на единицу каждый раз при выполнении определенной операции. Например, можно создать на Web-странице счетчик щелчков на кнопке или на гиперссылке. Часто счетчики применяются в циклах. Создать счетчик очень просто:

```
// Начальное значение счетчика
var count = 0;
... конструкция с оператором цикла или код функции обработки события
// инкрементация счетчика
count = count + 1;
```

Чтобы упростить запись, в JavaScript используются операторы инкремента:

count+ + ; – увеличивает значение переменной count на единицу;

count – –; – уменьшает значение переменной count на единицу.

В данном случае не важно, находится оператор инкремента до или после переменной. Конструкции ++count и count++ дадут одинаковый результат. Но положение оператора инкремента становится важным, если он используется в сочетании с оператором присваивания. Рассмотрим следующие примеры:

```
num1 = ++count; // значение count увеличивается на 1
// и присваивается переменной num1
num2 = count++; // значение count присваивается переменной
// num2, после чего увеличивается на 1
```

Объект Math

Объект Math поддерживается всеми обозревателями, поэтому можно смело использовать коллекцию методов и констант этого объекта для выполнения вычислений. При этом не нужно создавать объект Math или добавлять его в сценарии, он всегда доступен по умолчанию. Например, чтобы найти квад-

ратный корень от числа в переменной fVal, выполните следующее: var result = Math.sqrt(fVal);

Польза от применения объекта Math очевидна уже по этому примеру, так как JavaScript не содержит операторов для извлечения квадратного корня, а написать самостоятельно такую программу будет весьма затруднительно. Давайте рассмотрим ресурсы объекта Math.

Константы

В вычислениях нам часто приходится использовать константы, например число *к* (пи) или число Авогадро (количество молекул в одном моле вещества). Первую константу вы найдете в объекте Math, вторую, к сожалению, нет. Список констант небольшой, но полезный (табл. 1.3).

Таблица 1.3

Математические константы объекта Math

Константа	Описание
E	Число e – основание натурального логарифма (≈ 2,72)
LN10	Число ln(10) ≈ 2,3
LN2	Число ln(2) ≈ 0,69
LOG10E	Число lg(e) ≈ 0,43
LOG2E	Число $\log_2(e)$ ≈ 1,44
PI	Число π ≈ 3,14
SQRT1_2	Квадратный корень от 1/2 ≈ 0,71
SQRT2	Квадратный корень от 2 ≈ 1,41

Методы

Коллекция методов, предназначенных для выполнения математических операций, представлена в табл. 1.4.

Таблица 1.4

Методы объекта Math

Метод	Описание
Тригонометрические методы	
abs (x)	Возвращает абсолютное (положительное) значение от числа *x*
acos(x)	Арккосинус числа x (-1 <x<1)
asin(x)	Арксинус числа x (-1<x<1)
atan (x)	Арктангенс числа x
cos(x)	Косинус числа x
sin(x)	Синус числа x
tan(x)	Тангенс числа x

Продолжение таблицы 1.4

Методы округления	
ceil (x)	Возвращает ближайшее целое число, которое меньше или равно x
floor (x)	–"– ближайшее целое число, которое больше или равно x
round (x)	–"– целое число, ближайшее к x (Math.round(1.5) = 2)
Методы сравнения	
max(x,y)	–"– большее из двух чисел
mix(x,y)	–"– меньшее из двух чисел
Методы вычислений	
exp(x)	Возвращает e^x
log(x)	–"– натуральный логарифм *ln(x)*
pow(x,y)	–"– x^y
sqrt(x)	–"– квадратный корень от x

Генератор случайных чисел

Объект Math содержит *рандомайзер* – функцию, возвращающую случайное число. Вызов генератора случайных чисел выполняется следующим образом:

```
var rand = Math.random();
```

Переменной rand будет присвоено случайное число с плавающей запятой в диапазоне от 0 до 1. Используя различные ухищрения, можно получить числа в любом диапазоне. Например, ниже показана программная модель игральной кости, возвращающей целое число от. 1 до 6:

```
var rand = Math.floor(5.0*Math.random() + 0.5);
```

Операции присваивания

Оператор присваивания = уже использовался нами ранее во многих листингах. Но в JavaScript есть ряд дополнительных операторов присваивания, полезных в тех случаях, когда переменной присваивается новое значение исходя из текущего, т.е. одна и та же переменная находится слева и справа от оператора присваивания, как в следующем примере: num = num + 5;

Это выражение можно упростить: num += 5;

Полный список операторов присваивания показан в табл. 1.5.

Таблица 1.5

Операторы присваивания

Оператор	Описание
x = y	Присваивает переменной x значение переменной y
x += y	Увеличивает значение переменной xна значение переменной y
x -= y	Уменьшает значение переменной xна значение переменной y
x *= y	Увеличивает значение переменной x в y раз
x /= y	Уменьшает значение переменной x в y раз
x %= y	Присваивает переменной x остаток отделения значения переменной x на значение переменной y

Операции сравнения

В том случае, если нужно выяснить отношения равенства или отличий значений в двух переменных, применяйте операторы сравнения:

== – равенство;
! = – неравенство;
> – больше;
>= – больше или равно;
< – меньше;
<= – меньше или равно.

Все операторы сравнения возвращают логические значения true (истинно) и false (ложно). Например, в выражении

```
var bVal = 5 == 3;
```

переменной bVal будет присвоено значение false. Чаще всего операторы сравнения используются для проверки условия в конструкции с оператором if.

В синтаксисе операторов сравнения часто возникают ошибки. Обратите внимание на то, что оператор равенства содержит два символа равенства (==), тогда как один символ равенства имеет в JavaScript совершенно иной смысл (см. выше раздел «Операции присваивания»). Следует также помнить, что в операторах неравенства, больше или равно и меньше или равно символ равенства находится справа. Комбинации символов =!, => и =< вызовут ошибку.

Строковые операции

Со строками текста на Web-страницах приходится работать даже чаще, чем с числовыми значениями. В связи с этим в JavaScript предусмотрен обширный набор средств для анализа текста и манипуляций со строками. Как вы увидите, к строковым переменным можно применять даже некоторые математические операторы, с которыми вы познакомились выше. Набор удобных методов для работы с текстом предоставляет объект String.

Конкатенация строк и полиморфизм операторов

Чаще всего в сценариях приходится объединять фрагменты текста, хранящиеся в разных переменных, в одну фразу для вывода ее в окне обозревателя. Процесс объединения строк называется *конкатенацией.* Проще всего это сделать с помощью оператора суммирования «+», с которым вы познакомились выше, в разделе «Выполнение базовых математических операций». Рассмотрим пример в листинге 1.5.

Листинг 1.5

Конкатенация строк с помощью оператора суммирования

```
<SCRIPT>
Str1 = "Глубокоуважаемый ";
str2 = prompt("Введите имя и фамилию");
if (str2)
document.write(str1  + str2  +   "!");
</SCRIPT>
```

Сценарий выводит приветственную строку, которую можно использовать в начале делового письма. Сначала создается строковая переменная str1 с текстом "Глубокоуважаемый". Затем нужно ввести имя и фамилию человека, к которому мы обращаемся. Программа открывает диалоговое окно с просьбой к пользователю ввести требуемую информацию. Введенная строка сохраняется в переменной str2. Далее программа выводит текст на экран с помощью метода write, в аргументе которого происходит суммирование строк. Результат показан на рис. 1.8.

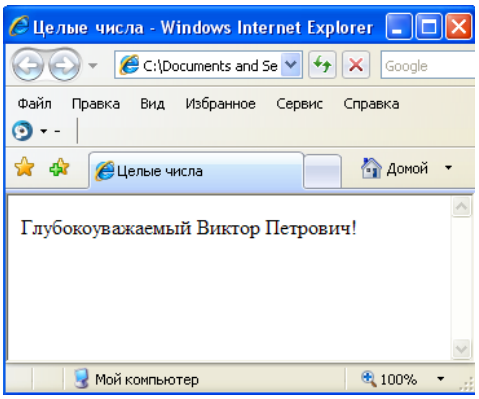

Рис. 1.8*. Результат конкатенации строк*

Для конкатенации строк используется оператор суммирования, который ранее применяли для выполнения сложения двух числовых переменных. Такое свойство операторов, при котором результат их выполнения зависит от типа переменных, к которым они применяются, называется *полиморфизмом* операто-

ров. К полиморфизму также можно отнести автоматическое изменение оператором типа одной из переменных, чтобы избежать возникновения конфликта. Рассмотрим следующее выражение: val = 3 + "2";

Как вы думаете, какой результат будет занесен в переменную val? Возможно, вас удивит, когда функция alert (val) выведет в окне сообщения значение 32. Все дело в том, что если одна из переменных оператора суммирования окажется строкой текста, даже при условии, что она содержит цифру, вторая переменная автоматически преобразуется в строку и выполняется конкатенация строк. Такой же результат получится в выражении.

```
val = 3;
val += "2";
```

Все остальные математические операторы, -, *, /, %, действуют в противоположном направлении: пытаются преобразовать строку в число. Если это не удается, выражение возвращает значение NaN. Рассмотрим следующие примеры:

```
val = 3 – "2"; // в результате получаем 1,
val = "s" * 3; // в результате получаем NaN.
```

Объект String

Все элементы текста Web-страницы и текстовые переменные, включая содержимое текстовых полей, являются экземплярами (объектами) класса String. В связи с этим к методам класса String можно обращаться напрямую: *строковая_переменная. метод (аргументы).* Все объекты класса String содержат переменную-член length, в которой хранится информация о длине строки – числе символов, включая пробелы. Попробуйте ввести следующий код:

```
sVal = "Привет!"; alert(sVal.length);
```

В данном случае сценарий покажет в окне сообщения число 7. Объекты String содержат много методов, полезных для работы со строками текста, часть которых представлена в табл. 1.6.

Таблица 1.6

Методы объекта string

Метод	Описание
Методы манипуляций со строками	
concat(str2, str3, ...)	Последовательно конкатенирует строки в списке аргументов со строкой, к которой был вызван метод strl.concat(str2, str3, str4);

split (*"разделитель"*)	Разбивает строку на массив подстрок по позициям заданного символа разделителя.
	sVal = "А,Б,В,Г";
	sArray = sVal. split (",");
	document.write(sArray[0]); // Выведет букву "А"
	document.write(sArrayfl]); // Выведет букву " Б"
	document. wr i t e (sArray. 1 ength) ; // Выведет 4

substr (позиция, длина)	Возвращает подстроку, заданную позицией и длиной. Отсчет позиций начинается с 0
substring (позиция, позиция)	Возвращает подстроку, заданную позициями начала и конца подстроки в строке. sVal = "абвгде"; alert(sVal.substring(1,3)) ; // возвратит "бв"
Методы поиска	
charAt (позиция)	Возвращает символ по указанной позиции в строке
indexOf (*"подстрока"*)	Возвращает позицию первого вхождения подстроки в строку
lastIndexOf (*"подстрока"*)	Возвращает позицию последнего вхождения подстроки в строку
Методы форматирования	
big ()	Увеличивает размер шрифта на единицу (см. ниже метод fontsize)
bold ()	Устанавливает полужирный шрифт
fontcolor(‘цвет')	Устанавливает цвет строки
fontsize('n')	Устанавливает номер размера строки от 1 до 7
Italics()	Задает курсив
link('URL')	Создает из строки гиперссылку на заданный URL-адрес или команды 'mailto:' и 'javascript:'
small ()	Уменьшает размер шрифта на единицу
strike()	Перечеркивает текст

Все перечисленные выше методы форматирования текста выполняются путем заключения строки в соответствующие дескрипторы HTML.

В листинге 1.6 рассмотрим работу функции find_month, в которой используются методы объекта String.

Листинг 1.6

Функция find_month

```
function find_month()  {
month  =  prompt("Введите месяц:   ");
if  (month)  {
var  i  = NaN;
list  = months.join("$");
position =  list.indexOf(month);
if  (position >  -1)  {
var  index =   0;
if  (position)  {
sub_list  =  list.substring(0,position);
sub_array =  sub_list.split("$");
index =  sub_array.length  –  1;
 }
alert("Месяц  "  + months[index]  +  "  в массиве находится под
индексом  "  +  index);
}
else
alert("Месяц  "  + month +  "  в массиве отсутствует")
}
}
```

С помощью функции find_month необходимо определить индекс текстового элемента в массиве. Для этого мы преобразовали массив в строку с помощью метода join, установив в качестве разделителя символ доллара ($). (Мы специально выбрали символ, который не ожидали увидеть в строках нашего текста.) Затем с помощью метода indexOf определили позицию вхождения искомого слова в строке текста. Если искомое слово не обнаружено, метод indexOf возвращает значение -1. Убедившись с помощью оператора if, что поиск завершился успехом, мы вырезаем часть строки от начала до позиции искомого слова, воспользовавшись для этого методом substring. Полученную строку вновь преобразуем в массив с помощью метода split, установив в качестве разделителя все тот же символ доллара. Нетрудно понять, что индекс искомого элемента в исходном массиве будет на единицу меньше размера нового урезанного массива.

Рассмотрим теперь пример форматирования текста с помощью методов объекта String (листинг 1.7).

Листинг 1.7

Изменение размера текста

```
<SCRIPT>
var sVal = "Установка размера шрифта";
for (i=1; i<=7; i++)
document.write(sVal.fontsize(i)  +  "<BR>");
</SCRIPT>
```

В листинге 1.7 мы используем метод fontsize для установки размера шрифта. Размер шрифта задается номером от 1 до 7. Как будет выглядеть текст на экране, зависит не только от номера шрифта, но и от установок параметров обозревателя. В листинге 2.6 мы выводим один и тот же текст с размером шрифта от 1 до 7, воспользовавшись оператором цикла for.

Результат показан на рис. 1.9.

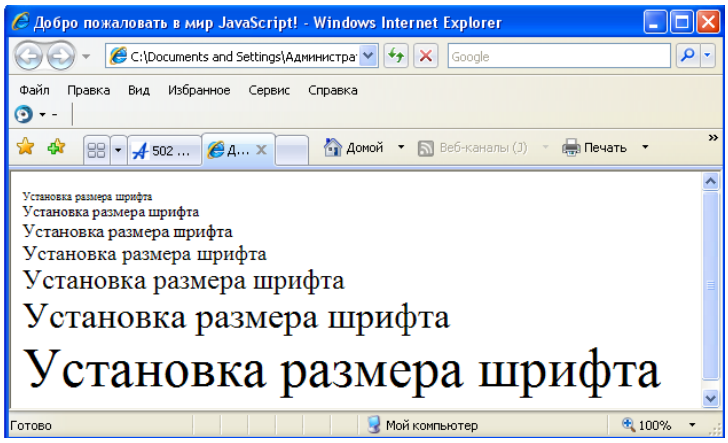

Рис. 1.9. *Изменение размера шрифта с помощью метода fontsize*

Методы форматирования объекта string добавляют дескрипторы в строку текста только в том случае, если текст выводится на экран с помощью метода write. Если просто применить метод форматирования к строковой переменной, результат не сохранится и никак не повлияет на формат текста при последующем выводе значения переменной на экран.

Вышеуказанная проблема затрудняет установку для строки сразу нескольких параметров форматирования, например размера шрифта и курсива.

Эту проблему можно решить путем добавления необходимых дескрипторов непосредственно в строку текста. Например, добавим в листинг 1.7 следующую строку кода где-нибудь после создания переменной sVal, но до цикла for:

```
sVal = "<I>"  +  sVal  +  "</I>";
```

Результат показан на рис. 1.10.

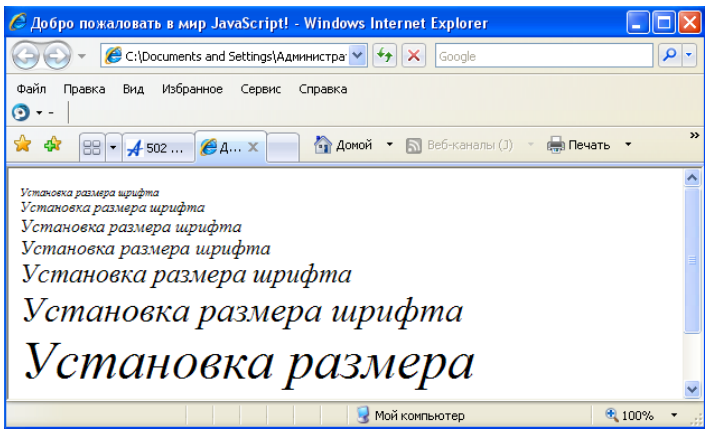

***Рис. 1.10**. Применение к тексту нескольких параметров форматирования*

1.9. Управление ходом выполнения сценария

В этом разделе вы познакомитесь с операторами, применяемыми в JavaScript для управления ходом выполнения программы.

Сравнение значений

Чтобы установить момент выполнения условия, нужно сравнить текущее значение переменной со значением, характеризующим данное условие. Выбор переменной и стандартного значения зависит исключительно от логики процесса, который вы хотите смоделировать с помощью сценария. Например, таким условием может быть значение переменной-счетчика. Если доступ к странице заблокирован с помощью пароля, можно установить счетчик попыток пользователя ввести пароль. Контрольное значение счетчика может равняться трем, пяти, во всяком случае меньше десяти, после чего программа блокирует доступ к серверу для этого пользователя.

Условием выполнения программы может быть значение, возвращенное диалоговым окном. Если пользователь щелкнет на кнопке **Отменить**, функция диалогового окна возвратит значение false. Щелчок на кнопке OK возвратит значение true или текст, введенный пользователем в диалоговое окно. Условием может быть соответствие определенному формату значения, введенного в текстовое поле диалогового окна и формы.

Другим часто используемым условием может быть наличие или отсутствие на Web-странице определенного элемента. Отсутствию элемента Web-страницы соответствует значение NaN. Это же значение возвращается некоторыми функциями в случае возникновения в них ошибки.

В этом разделе мы рассмотрим способы определения соответствия значения переменной определенному критерию.

Что такое «истинно» и «ложно»

В JavaScript используются шесть операторов, = =, ! =, <, <=, > и >=, с помощью которых можно установить равенство, неравенство или превышение значения одной переменной относительно другой. Общим для всех этих операторов является то, что все они возвращают значения логического типа true (истинно) и false (ложно). Смысл этих значений очевиден: выражение 5 = = 5 возвратит true, тогда как выражение 5 = = 2 возвратит false.

Переменные логического типа тесно связаны с переменными других типов и могут быть заменены ими в логических выражениях. Так, числовое значение 0 соответствует логическому false, в то время как все другие числа соответствуют значению true. Если переменная содержит в себе объект (элемент Web-страницы или текст), то в логическом выражении эта переменная будет соответствовать значению true, тогда как значение NaN и пустая строка текста "" будут рассматриваться как false.

С логическими переменными работают условный оператор if и операторы циклов. Если условная переменная принимает значение true, то оператор if запускает на выполнение код одной подпрограммы, а если она принимает значение false – другой. Циклы выполняются до тех пор, пока условная переменная имеет значение true.

Конструирование логических выражений

С помощью оператора сравнения можно проверить значение одной переменной. Но иногда бывает необходимо в качестве условия установить попадание значения переменной в определенный диапазон или связать выполнение подпрограммы с несколькими условиями. В подобных случаях нужно применять логические выражения. Для создания таких выражений используются специальные логические операторы && (логическое *и)* и | | (логическое *или).*

С помощью оператора && можно определить одновременное выполнение двух условий, как в следующих примерах:

```
bVal = (x >= 20) && (x <= 30) // возвращает true, если x
                              // находится в диапазоне [20:30]
bVal = (x>20) && y // возвращает true, если x больше 20
```

```
                    //и значение переменной y соответствует true
```

Обратите внимание на то, что вместо выражения сравнения y = = true можно использовать только имя самой переменной y.

В случае применения оператора | | выражение возвратит true, если хотя бы одно из двух условий выражения будет истинным. Например, выражение

```
bVal = (txtField.value != "") | | (MyArray.length > 0)
```

возвращает true, если текстовое поле txtField содержит непустую строку текста или массив MyArray содержит хотя бы один элемент.

Логические выражения могут быть еще более сложными и содержать несколько логических операторов, например, как следующее:

```
bVal = ((x >= 20) && (x <= 30)) | | (y == x)
```

возвращает true, если x находится в диапазоне [20:30], или значение y равно значению x.

Выполнение задач по циклу

Если нам необходимо выполнить одну и ту же операцию с разными объектами или с разными значениями переменной, проще всего это сделать, организовав выполнение части кода программы по циклу. Для организации циклов в JavaScript используются два оператора: for и while. Помимо синтаксиса, разница между этими операторами состоит в логике организации цикла. Оператор for лучше использовать в тех случаях, когда цикл нужно выполнить определенное число раз, а оператор while лучше подходит для циклов, повторяющихся до достижения определенного условия или для создания бесконечных циклов.

Цикл for

Для организации цикла с помощью оператора for используется следующий синтаксис:

for *(счетчик = начальное_значение; условие_завершения; приращение_счетчика*)

{ *тело цикла* }

За оператором for следует код *определения цикла*, заключенный в круглые скобки, и заключенный в фигурные скобки код *тела цикла* – программа, повторяемая по циклу.

Если тело цикла for состоит всего из одной строки кода, то фигурные скобки можно не устанавливать. Это справедливо также для исполняемого кода операторов while, if и else.

Определение цикла состоит из трех выражений, разделенных символами точки с запятой:

- первое выражение создает целочисленную переменную-счетчик и присваивает ей начальное значение;
- второе выражение задает условие, при невыполнении которого цикл прерывается;
- третье выражение устанавливает способ приращения счетчика. Чаще всего это *счетчик++,* что означает приращение счетчика на единицу после каждого цикла. Но можно использовать и более сложные выражения, например: счетчик = счетчик – 5.

Одна из часто встречаемых задач, где используется цикл for, – это заполнение данными массивов и словарей или же, наоборот, вывод данных из массивов и словарей на экран. Давайте рассмотрим пример выполнения обеих этих задач в листинге 1.8.

Листинг 1.8

Заполнение массива и вывод элементов на экран с помощью цикла for

```
<SCRIPT>
var size = parseInt(prompt("Введите размер массива","5"));
var ar = new Array(size);
for (n = 0; n < ar.length; n++) {
sVal = prompt("Введите текст элемента массива","");
ar[n] = sVal;
document.write(ar[n] + "<BR>");
}
</SCRIPT>
```

Сначала программа показывает диалоговое окно с предложением ввести размер массива (по умолчанию предлагается значение 5). Значение размера преобразуется в целое число с помощью стандартной функции parseInt и присваивается переменной size. В следующей строке программа создает новый массив ar указанного размера. Затем следует предмет нашего изучения – цикл for. В определении i цикла создается переменная-счетчик n, которой присваивается начальное значение 0. Цикл будет выполняться до тех пор, пока значение n

меньше размера созданного массива ar, и n увеличивается на единицу после выполнения каждого цикла.

Тело цикла содержит три строки (поэтому заключено в фигурные скобки). Сначала программа предлагает пользователю ввести текстовое значение для текущего элемента массива. В следующей строке мы воспользовались счетчиком n как индексом массива для последовательного доступа ко всем его элементам. Текущему элементу присваивается значение, введенное пользователем. Последняя строка тела цикла выводит значение текущего элемента массива на экран с помощью метода document .write.

Промежуточный этап заполнения массива показан на рис. 1.11.

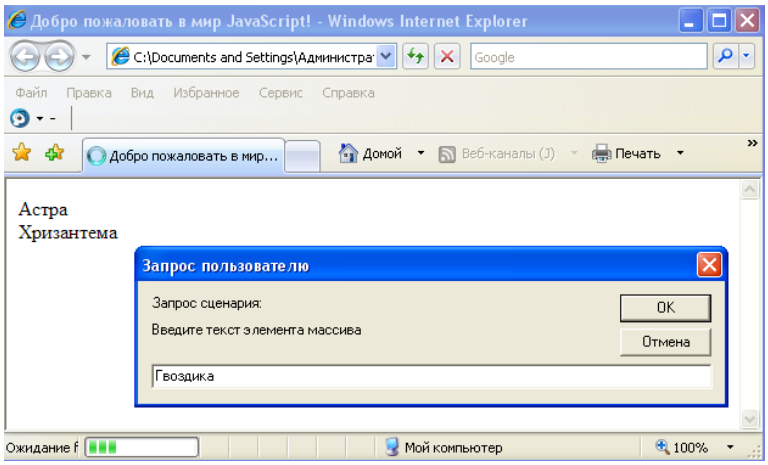

***Рис. 1.11**. Заполнение массива с помощью цикла for*

Цикл while

Цикл while, в отличие от цикла for, не содержит счетчика и длится до тех пор, пока выполняется заданное логическое условие. Как только логическое выражение, установленное в определении цикла, возвратит false, цикл прекратится. Это условие может никогда не наступить, тогда цикл продолжается бесконечно. Возникновение бесконечного цикла обычно вызывает ошибку в выполнении программы и зависание приложения. Но иногда бесконечные циклы создаются целенаправленно. Ниже вы узнаете об использовании оператора break для прерывания цикла, что позволяет сделать цикл зависимым от нескольких условий.

Синтаксис цикла while:

while *(логическое_выражение)* { *тело цикла* }

В листинге 1.9 показан пример, в котором раскрывается одна интересная особенность цикла while.

Листинг 1.9

Особенности проверки условия в цикле while

```
<SCRIPT>
count = 1;
while (count< 5) {
document.write("Счетчик " + count + "<BR>");
count++; )
document .write ( "<BRxBR>") ; count = 0; while (count< 5) {
count++;
document.write("Счетчик  "  + count +  "<BR>"); }
</SCRIPT>
```

В сценарии используются два, на первый взгляд, одинаковых цикла while. В условии проверяется значение счетчика, который инкрементируется при каждом выполнении тела цикла. Метод write в теле обоих циклов выводит на страницу текущее значение счетчика. Разница только в том, что в первом цикле приращение счетчика происходит в последней строке тела цикла, а во втором цикле это первая строка. Начальные значения счетчика были подобраны таким образом, чтобы первым значением, выводимым на экран, была единица. Несмотря на то, что условия циклов были одинаковыми, второй цикл вывел значения от 1 до 5, а первый – от 1 до 4. Все дело в том, что во втором цикле приращение счетчика происходило сразу после проверки условия, тогда как в первом – непосредственно перед проверкой.

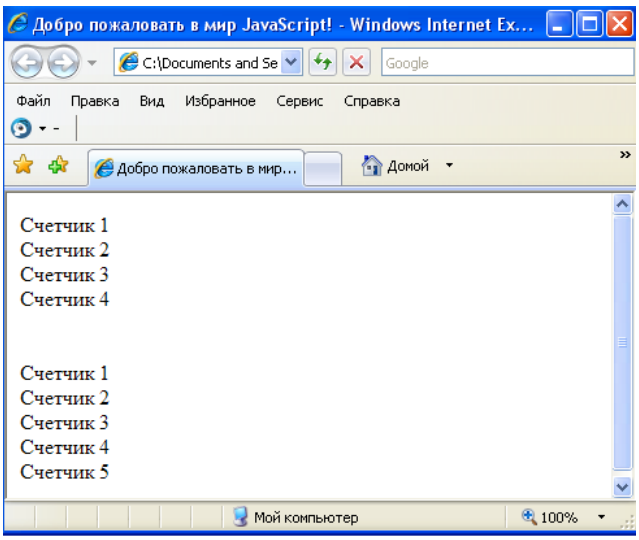

Рис. *1.12. Особенности проверки условия в цикле while*

Вложенные циклы

В сценариях иногда используются конструкции, когда один цикл создается в теле другого цикла. Такие циклы называются *вложенными*. Нет никаких ограничений на использование вложенных циклов и число вложений, кроме логики и здравого смысла. Это означает, что цикл for может быть вложен в тело другого цикла for или цикла while и иметь в своем теле любой другой цикл.

Обычно вложенные циклы используют для математической обработки многомерных массивов данных или для вывода данных на экран в табличной форме. Причем, манипулируя условиями циклов, можно создать довольно сложные композиции, как в примере на рис. 1.13.

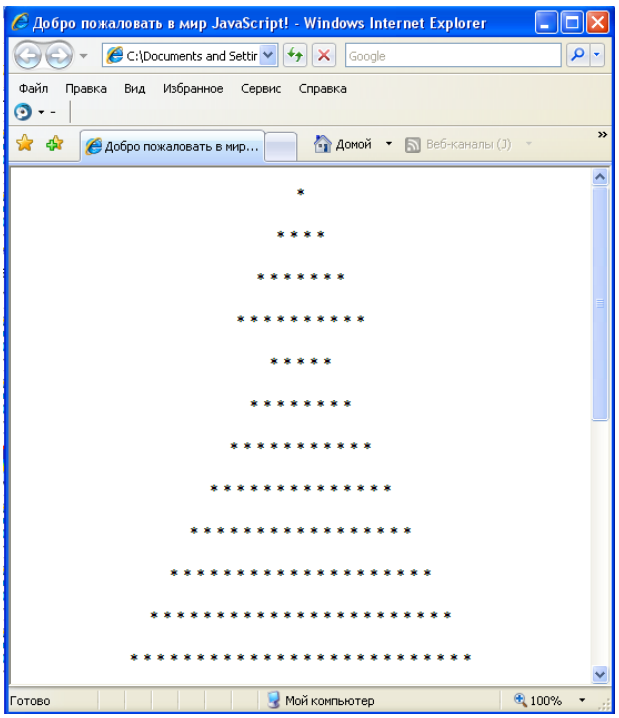

Рис. *1.13. Елочка создана с помощью трех вложенных циклов for*

Код сценария, с помощью которого была создана композиция на рис. 1.13, показан в листинге 1.10.

Листинг 1.10

Создание двухмерной композиции с помощью вложенных циклов

```
<SCRIPT>
for (i = 1;i < 10;i += 4) {
for (j = 0; j < i + 3; j++) {
document.write("<CENTER>") ;
for (n = 0; n < i + j*3; n++)
document.write("* ");
document .write ("</CENTER><BR>") ;
}
}
</SCRIPT>
```

Первый цикл for со счетчиком i устанавливает число уровней елочки и число символов звездочки в вершине каждого уровня. Следующий цикл со счетчиком j определяет число строк в каждом уровне, а последующий вложенный цикл со счетчиком n устанавливает количество звездочек в каждой строке в зависимости от уровня. Взаимодействие вложенных циклов достигается за счет того, что в условии каждого вложенного цикла используются значения счетчиков цикла верхнего уровня. Обратите также внимание на то, как нам удалось выровнять по центру звездочки в окне обозревателя с помощью дескриптора <CENTER>.

Приведем другой, более практичный, пример использования вложенного цикла для динамического вывода данных многомерного массива в таблице. Код сценария показан в листинге 1.11.

Листинг 1.11

Динамическое создание таблицы по данным многомерного массива

```
<BODY>
<TABLE NAME='tb' ID='tb' BORDER=1></TABLE>
<SCRIPT>
arData = new Array(3);
arData[0] = new Array("1","2","3","4","5");
arData[1] = new Array("а","б","в","г","д");
arData[ 2 ] = new Array("@","#","$","%","&");
for (i = 0;i < arData.length;i++) {
tb.insertRow(i);
for (j = 0; j < arData[i].length; j++) {
tb.rows[i].insertCell(j ) ;
tb.rows[i].cells[j].width = 50;
tb.rows[i].cells[j].align = 'center';
tb.rows[i].cells[j].innerText = arData[i][j];
}
}
</SCRIPT>
</BODY>
```

В основной раздел Web-страницы добавляется пустая таблица под именем tb. Затем выполняется код сценария, в котором прежде всего создается и заполняется данными многомерный массив arData. В следующих строках сценария выполняется конструкция из двух вложенных циклов for. Первый цикл добавляет в таблицу новую строку, тогда как следующий цикл заполняет строку таблицы ячейками и присваивает им значения из массива.

Обратите внимание на то, как счетчики циклов используются для адресации ячеек таблицы и элементов многомерного массива. Полученная таблица показана на рис. 1.14.

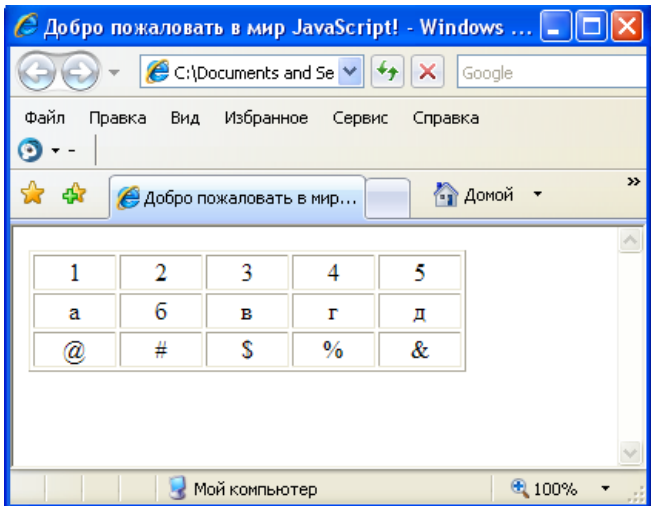

***Рис. 1.14**. Вывод данных многомерного массива с помощью конструкции вложенных циклов*

Прерывание и продолжение цикла

В JavaScript есть специальные команды, с помощью которых можно задействовать дополнительные уровни контроля за выполнением циклов:

- **break** – прерывает цикл в любой точке тела цикла;
- **continue** – заставляет программу пропустить все следующие строки тела цикла и начать новый цикл.

Эти команды используются внутри циклов в конструкции с операторами if-else, чтобы проверить дополнительные условия выполнения цикла.

Команда **break** является необходимым элементом бесконечных циклов.

Рассмотрим пример. Предположим, вы хотите защитить свою страницу с помощью пароля и сделать ее доступной только для тех, кто умеет умножать числа. Введите код листинга 1.12.

Листинг 1.12

Прерывание цикла с помощью команды break

```
<SCRIPT>
var c = 1;
while(c) {
num1 = Math.round(Math.random()*19 + 1);
num2 = Math.round(Math.random()*19 + 1);
result = prompt(("Введите произведение " + num1 + " x " +
num2),"");
if (result && parseInt(result)) {
if (result == num1*num2) break
}
}
</SCRIPT>
```

В начале сценария запускается бесконечный цикл while. Для этого переменной ***c*** присваивается значение 1, а в условии оператора проверяется, являет-

ся ли значение ***c*** истинным. Поскольку все ненулевые значения истинны, а переменная ***c*** нигде не изменяется, цикл while будет продолжаться вечно, если его не остановить с помощью команды break. Далее программа генерирует два случайных целых числа в диапазоне [1:20] и предлагает пользователю ввести результат произведения этих чисел. В следующих строках с помощью двух операторов if программа проверяет, является ли введенное значение числом и соответствует ли оно произведению. Если да, цикл прерывается и продолжается загрузка Web-страницы. Если результат неправильный или пользователь щелкнул на кнопке **Отмена**, цикл будет повторяться вновь и вновь. Промежуточный этап выполнения программы показан на рис. 1.15.

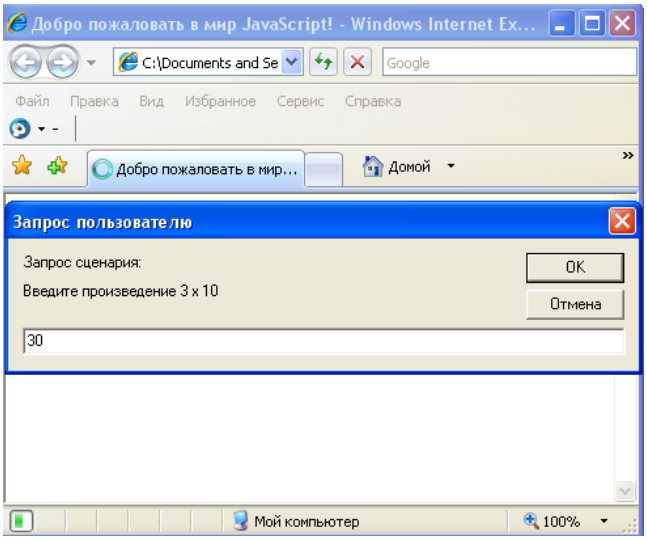

Рис. 1.15. *Ввод числа согласно запросу*

Команду **continue** часто применяют в случаях, когда необходимо отследить недопустимое значение счетчика или другой изменяемой переменной цикла, пропустить выполнение цикла для этой переменной, но продолжить цикл для последующих значений. Например, предположим, что мы хотим вывести обратные значения для ряда целых чисел. Для любого числа *x* можно получить значение *1/x* за единственным исключением – число нуль. Деление на нуль вызовет ошибку. Поэтому в программе следует установить проверку равенства переменной нулю, прежде чем выполнять деление. Рассмотрим пример в листинге 1.13.

Листинг 1.13

Прокручивание цикла вхолостую с помощью команды continue

```
<HTML>
<HEAD>
<SCRIPT>
function culc() {
var start = Math.round(document.forms[0].elements[0].value);
var end = Math.round(document.forms[0].elements[1].value);
if (start && end && (start < end)) {
output.innerHTML = "";
for(n = start;n <= end;n++) {
if (n == 0) continue sVal = 1/n + ";<BR>";
output.innerHTML += sVal;
}
}
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
Первый член ряда: <INPUT TYPE='text' VALUE='-5 ' ><BR> По-
следний член ряда: <INPUT TYPE='text' VALUE= ' 5 ' ><BR>
<INPUT TYPE='button' VALUE='Вывести ряд обратных чисел'
ONCLICK='culc() '><BR>
</FORM>
<P NAME= ' output' ID=' output '></P>
</BODY>
</HTML>
```

Web-страница предлагает ввести в поля формы значения начала и окончания ряда целых чисел, для которого программа рассчитает и выведет обратные значения *1/x,* как показано на рис. 1.16.

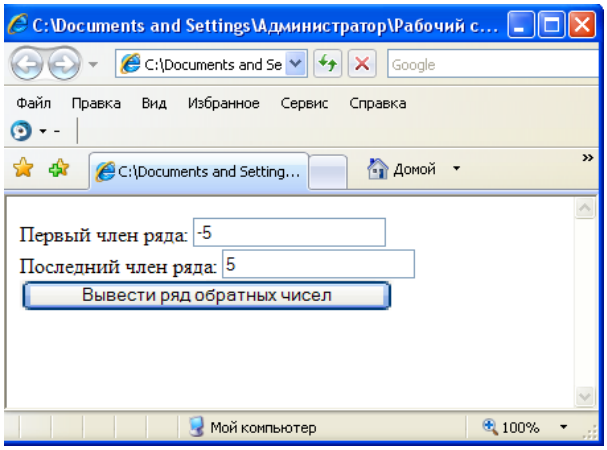

Рис. 1.16. Программа рассчитывает ряд чисел

Выполнение сценария запускается щелчком на кнопке **Вывести ряд обратных чисел**. Обратите внимание на то, как в сценарии мы можем обратиться к значениям полей, даже если им не были присвоены уникальные имена. Для этого воспользуемся массивами элементов, о которых вы узнали в гл. 2.

Далее программа проверяет, были ли введены какие-либо данные в поля, а также следит за тем, чтобы первый член числового ряда был меньше последнего члена. Затем очищается текст именованного абзаца и запускается цикл for, в котором вычисляются значения *1/x*. Если в числовом ряду оказывается нуль, то выполняется команда continue, которая запускает следующий шаг цикла со значением счетчика, увеличенным на единицу.

Ветвление программного кода

Циклы делают программный код мощнее, позволяя многократно использовать один и тот же фрагмент кода для последовательной обработки множества значений в массиве данных. Благодаря ветвлению программного кода сценарий становится более гибким, чувствительным к контексту выполнения программы и к требованиям пользователя. Выбор той или иной подпрограммы зависит от выполнения или невыполнения определенного условия.

Оператор if

Оператор if наиболее часто используется для ветвления программного кода или выполнения части кода по условию. В простейшем виде синтаксис оператора if выглядит так:

if (*логическое выражение*)

{ *код, выполняемый в случае возвращения логическим выражением значения true}*

Если логическое выражение в операторе if возвратит false, весь блок кода, заключенный в фигурные скобки, будет пропущен, и управление перейдет к строке сценария, следующей за блоком оператора if.

Если код, находящийся под управлением оператора if, состоит всего из одной строки, фигурные скобки можно не вводить.

Таким образом, в первом варианте с помощью оператора if выделяется программный блок, выполнение которого необязательно и зависит от условия. Но зачастую необходимо создать два альтернативных программных блока, один из которых обязательно выполняется в зависимости от выполнения или

невыполнения условия. Для моделирования таких ситуаций используется конструкция операторов if-else:

if (*логическое выражение*)

{ *код, выполняемый в случае возвращения логическим выражением значения true*}

else

{ *код, выполняемый в случае возвращения логическим выражением значения false*}

Множественное ветвление программного кода

Если нужно сделать выбор между тремя или большим числом подпрограмм, одной конструкции if-else будет недостаточно. В зависимости от логики выполнения программы, проблему можно будет решить за счет вложенных конструкций if-else или с помощью операторов switch-case.

Вложенные операторы if

Часто ответ на один вопрос вызывает дополнительные вопросы. Логика таких сценариев следующая: если условие «A» выполняется, то нужно реализовать подпрограмму 1, если не выполняется, то проверить условие «B» и по результатам проверки запустить подпрограмму 2 или 3. Для решения этой проблемы можно использовать следующую комбинацию операторов if и else:

var A = *логическое выражение;*

var B = *логическое выражение;* if (A)

{ *код 1-й подпрограммы* }

else {

if (B)

{ *код 2-й подпрограммы* }

else

{ *код 3-й подпрограммы* }

}

Выбор из трех подпрограмм – это еще довольно простая задача. Часто в сценариях приходится проверять много разных условий, и бесконечные чередования if и else затрудняют чтение и понимание кода сценария. Чтобы упростить код, в JavaScript используется модифицированный вариант конструкции

if-else для проверки ряда условий: if-else-if. Предыдущая схема программного кода изменится следующим образом:

```
if   (A)
{ код 1-й подпрограммы }
else if (B){
{ код 2-й подпрограммы }
else
{  код 3-й подпрограммы }
}
```

Сначала проверяется условие «A». Если «A» – false, проверяется условие «B». Если «B» окажется false, можно таким же образом добавить еще проверку условия «C» и т.д. Конструкцию завершает одиночный оператор else, блок которого выполняется только в том случае, если все условия оказались false.

Рассмотрим применение этого подхода на примере. В листинге 1.14 представлен код сценария, который анализирует текстовый символ, введенный пользователем, и сообщает, является этот символ цифрой, буквой или знаком.

Листинг 1.14

Проверка типа символа с помощью вложенных операторов if

```
<SCRIPT>
var symbol = "" ;
while (symbol != null) {
symbol = prompt("Введите любой одиночный символ", symbol);
if {symbol == null)
break
else if((symbol>="a" && symbol<="z") || (symbol >= "A" && sym-
bol <= "Z"))
alert(symbol + " – буква латинского алфавита");
else if (symbol >= "A" && symbol <= "я")
alert(symbol + " – буква русского алфавита");
else if (symbol >= "0" && symbol <= "9")
alert(symbol + " – цифра");
else if (symbol == "")
alert("\"\" – пустая строка");
else
alert(symbol  +  "  –  знак препинания или специальный сим-
вол") }
</SCRIPT>
```

В сценарии запускается цикл while, в первой строке которого открывается диалоговое окно с просьбой ввести символ. Далее следует программа анализа введенного символа. Первый оператор if проверяет, не была ли нажата клавиша **Отмена**. (В этом случае функция диалогового окна возвратит объект null и оператор break прервет выполнение цикла.) Далее строки else if последовательно проверяют принадлежность символа к остальным группам: буквам латинского алфавита, русского алфавита, цифрам или пустой строке. Если символ не принадлежит ни к одной из этих групп, последний оператор else выводит сообщение, что был введен знак препинания или специальный символ.

Оператор switch

Оператор switch удобно применять для анализа переменных, которые могут принимать более двух значений. Типичный пример – обработка значений раскрывающегося списка с несколькими пунктами. Синтаксис оператора switch следующий:

switch *(выражение или переменная)* {
case *значение:*
строки кода
break;
case *значение:*
строки кода
break;
default;
строки кода break;
}

В условии оператора switch может быть имя переменной или выражение, возвращающее ряд значений. Тело оператора switch заключается в фигурные скобки и состоит из повторяющихся операторов case, проверяющих совпадение значения оператора switch с константным значением, установленным для каждого оператора case. Если совпадения не обнаружены, управление передается программному коду оператора default в конце блока оператора switch.

Обратите внимание на то, что блоки операторов case и default не выделяются фигурными скобками. Используется другой синтаксис: блок начинается

с символа двоеточия и завершается оператором break. 1.10. Упражнения на JavaScript

Упражнение 1. Ввод текста непосредственно на Web-странице

Введите в текстовом редакторе код Web-страницы, показанный в листинге 1.15.

Листинг 1.15

Ввод текста на Web-страницу

```
<BODY>
<FORM>
<INPUT TYPE=text  ID=txt_field NAME=txt_field SIZE=50><BR>
<INPUT TYPE=button VALUE='Ввести текст'
ONCLICK="insert_text(txt_field.value)">
</FORM>
<P  ID=par NAME=par></P>
<SCRIPT>
function  insert_text(text) {
par.innerText=text; }
</SCRIPT>
</BODY>
```

Web-страница содержит форму с текстовым полем txt_field и кнопкой **Ввести текст**. Событию ONCLICK этой кнопки назначено выполнение пользовательской функции insert_text, определенной ниже в разделе сценария. В функцию обработки события передается один аргумент – текущее значение текстового поля txt_field.value. Web-страница также содержит пустой именованный абзац: <P ID=par NAME=par></P>. Функция insert_text вставляет переданный ей текст в абзац: par. innerText=text;. Результат показан на рис. 1.17.

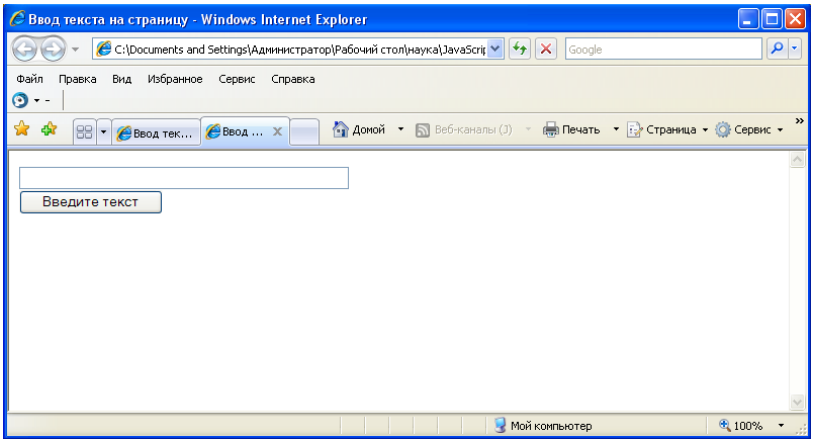

***Рис. 1.17**. Текст из поля был вставлен на Web-страницу щелчком на кнопке Ввести текст*

Упражнение 2. Всплывающий текст

Введите в текстовом редакторе код листинга 1.16.

Листинг 1.16

Добавление всплывающего текста

```
<HTML>
<BODY>
<HEAD>
<TITLE>Example</TITLE>
  <SCRIPT>
    function newContent() {
      alert("Загрузка нового контента");
      document.open();
      document.write("<h1>Долой старое, да здравствует новое!</h1>");
      document.close();
    }
  </SCRIPT>
</HEAD>

<body onload="newContent();">
  <p>Какой-то оригинальный контент.</p>
</BODY>
</HTML>
```

Откройте файл с помощью обозревателя. Обратите внимание на всплывающее окно, см. рис. 1.18.

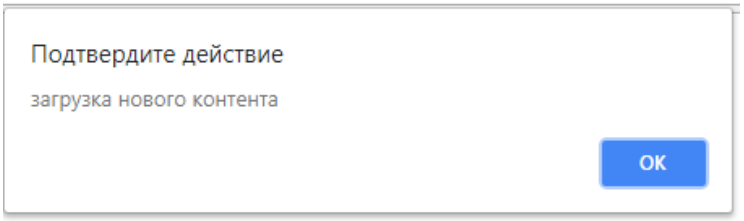

Рис. 1.18. *Щелкните на выделенном термине, чтобы увидеть его определение*

Упражнение 3. Меню гиперссылок

Если у вас на странице много гиперссылок, то для экономии места и облегчения поиска нужной ссылки можно сгруппировать их по категориям и добавить на страницу список категорий.

Введите в основной раздел страницы код листинга 1.17.

Листинг 1.17

Всплывающие списки гиперссылок

```
    <FORM><DIV STYLE=' color:  blue;  text-decoration:
underlined'>
    <P ONMOUSEOVER=f1()>группа 1</P>
    <UL  ID='group_1'   NAME='group_l'></UL>
    <P ONMOUSEOVER=f2()>группа 2</P>
    <UL ID='group_2'   NAME='group_2'></UL>
    <P ONMOUSE0VER=f3()>группа  3</P>
    <UL ID='group_3'   NAME='group_3'></UL>
    </DIV></FORM>

    <SCRIPT> function  f1()   {
    window.group_2.innerHTML=""
    window.group_3.innerHTML=""
    window.group_1.innerHTML="<LI  ONMOUSEOVER=on_select
('art1.htm')>Статья 1</LI><LI  ONMOUSEOVER=on_select
('art2.htm')>Статья 2</LI><LI  ONMOUSEOVER=on_select
('art3.htm')>Статья 3</LI>"
    };

    function  f2(){
    window.group_1. innerHTML=""
    window.group_3.innerHTML=""
    window.group_2.innerHTML="<LI  ONMOUSEOVER=on_select
('art4.htm')>Статья 4</LI><LI  onMouseOver=on_select
('art5.htm')>Статья  5</LI>"
    };
    function  f3(){
    window.group_1.innerHTML=""
    window.group_2.innerHTML=""
    window.group_3.innerHTML="<LI ONMOUSEOVER=on_select
    ('art6.htm')>Статья 6</LI>"
    };
    function on_select(name){
    top.main.location.replace(name);
    };
    </SCRIPT>
```

Разберем предложенный код HTML. Сначала создаются заголовки меню верхнего уровня *Группа 1 – Группа 3*, каждый в своем абзаце с установленными функциями обработки событий ONMOUSEOVER. За заголовками следуют объекты пустого маркированного списка с уникальным именем: <UL ID='group_#' NAME='group_l'></UL>. Программа работает следующим образом:

1. Пользователь наводит указатель мыши на заголовок группы, происходит событие ONMOUSEOVER, вызывающее на выполнение соответствующую функцию.

2. В первых строках функция обработки события обнуляет списки ссылок других категорий, присваивая пустые строки атрибутам innerHTML соответствующих маркированных списков.

3. В третьей строке функция создает список ссылок под выбранной категорией, присвоив HTML-код списка атрибуту innerHTML соответствующего маркированного списка.

4. При наведении указателя мыши на пункт всплывающего меню в основной рамке окна обозревателя отображается текст новой Web-страницы.

Результат работы представлен на рис.1.20:

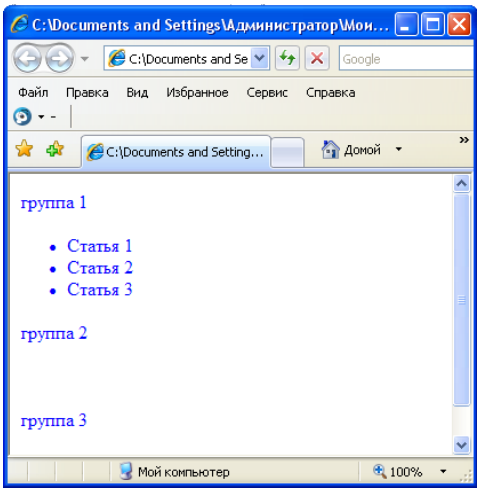

***Рис. 1.20**. Всплывающее меню гиперссылок*

Упражнение 4. Использование функции write

Листинг 1.18

Ввод текста Web-страницы с помощью функции write

```
<HTML>
<HEAD>
<TITLE>Функция write</TITLE>
<SCRIPT>
function input1() {
document.write("<P>Ввод текста с помощью пользовательской
функции.</P>");
}
function input2() {
win2 = window.open("","output","width=400,height=200");
win2.document.write("<P>Ввод текста в новое окно</P>");
win2.location.reload();
```

```
}
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
document.write("<P>Ввод текста из кода сценария.</P>")
input1()
document.write("<F0RMxINPUT TYPE=button VALUE='Вывести текст в новое окно' ONCLICK=input2()></FORM>")
</SCRIPT>
</BODY>
</HTML>
```

Отметим, что раздел <BODY> содержит только код сценария, который создает все элементы Web-страницы. Первая строка сценария добавляет текст «Ввод текста из кода сценария». Затем следует вызов функции *input1*, определенной в коде сценария в разделе заголовка. Несмотря на то, что команда write в этой функции записана в коде Web-страницы еще в разделе заголовка, на выполнение эта функция вызывается только сейчас. Поэтому текст «Ввод текста с помощью пользовательской функции» появится во втором абзаце в окне обозревателя. Затем, с помощью функции write, создается форма и командная кнопка **Вывести текст в новое окно**. Щелчок на этой кнопке запускает на выполнение функцию *input2*, которая открывает новое окно обозревателя и вводит в него строку текста.

Результат выполнения листинга 1.18 показан на рис. 1.21.

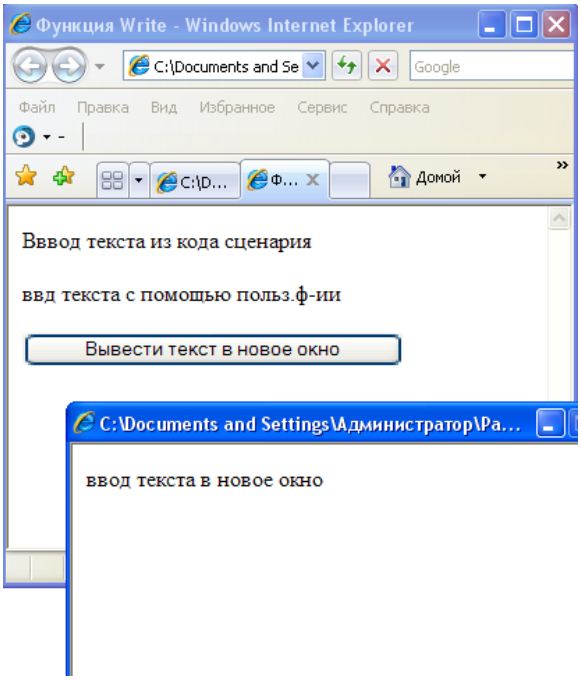

***Рис. 1.21**. Использование функции write для ввода данных в текущий документ и в новое окно обозревателя*

Упражнение 5. Динамическое редактирование таблиц

Для добавления строк и ячеек к таблице в JavaScript применяются специальные функции: insertRow*(номер_строки)* и insertCell *(номер_ячейки).* Удобство использования этих функций состоит в том, что они позволяют добавлять строки и ячейки в определенном месте таблицы.

Предположим, нам необходимо иметь возможность добавлять в таблицу на Web-странице новые записи с помощью полей формы, как показано на рис. 1.22.

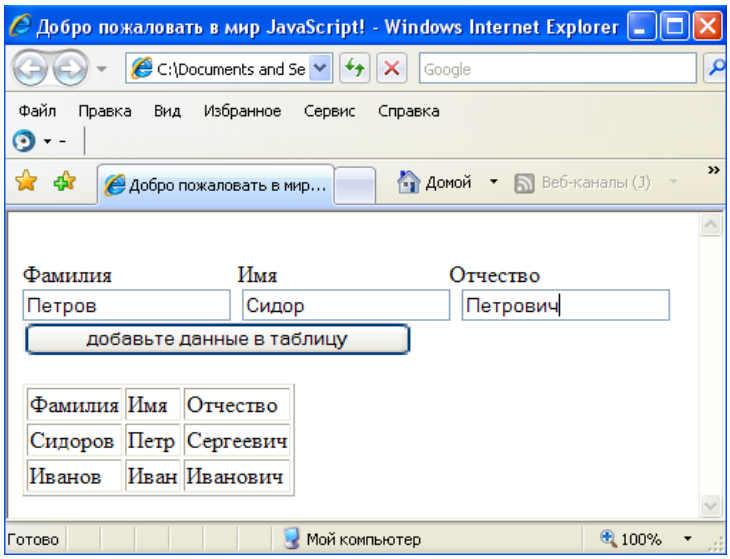

Рис. 1.22. Добавление записей в таблицу с помощью текстовых полей и кнопки

Сначала создается форма из трех текстовых полей. За формой в коде HTML была создана таблица под именем *table* со строкой заголовков, а кнопке **Добавить запись** в таблицу присвоена функция обработки события *add_entry*. Код сценария показан в листинге 1.19.

Листинг 1.19

Динамическое добавление строк в таблицу

```
<script>
function add_entry(surname, name, middle) {
row = table.insertRow(1);
cell3 = row.insertCell(0);
cell3.innerText = middle;
cell2 = row.insertCell(0);
cell2.innerText = name;
cell1 = row.insertCell(0);
cell1.innerText = surname;
}
```

```
    </script>

    <Form>
    <br>Фамилия
         &nb
sp; 
         Имя
         &nb
sp; 
             
        Отчество
    <BR>
    <INPUT Type='text' Value="Иванов" name='s' ID='s'> 
    <INPUT Type='text' Value="Иван" Name='n' ID='n'> 
    <INPUT Type='text' Value="Иванович" Name='m' ID='m'>
    <br><INPUT TYPE='button' Value='добавьте данные в таблицу'
ONCLICK = "add_entry(s.value,n.value,m.value)">
    </form>
    <table Name='table' ID='table' border=1>
    <tr><td>Фамилия</td><TD>Имя</td><TD>Отчество</td>
    </tr>
    </table>
```

В функцию передаются значения полей *Имя, Фамилия и Отчество*.

В первой строке функции add_entry в таблицу добавляется новая строка. Указан индекс ввода – 1. Таким образом, каждый щелчок на кнопке будет вставлять новую строку под строкой заголовка, смещая все остальные строки вниз.

Обратите внимание: нумерация строк, как и ячеек, начинается с числа 0, т.е. первая строка в таблице является нулевой. Номер последней строки равняется числу строк таблицы минус единица. Если индекс окажется больше этого числа, строка не будет добавлена в таблицу.

Затем происходит добавление ячеек в новую строку и ввод соответствующего текста. Все ячейки вводятся под индексом 0, и каждая новая ячейка смещает предыдущие вправо по строке. Поэтому первой добавленной ячейке мы присваиваем текст последней ячейки в строке. Попробуйте самостоятельно изменить код таким образом, чтобы ячейки вводились в строку последовательно, с первой по последнюю.

Упражнение 6. Динамическое изменение списков

Заменим на Web-странице таблицу объектом раскрывающегося списка <SELECT ID='selector' NAME='selector'></SELECT>. Заменим также код функции *add_entry*, как показано в листинге 1.20.

Листинг 1.20

Динамическое добавление пунктов в список

```
function add_entry(surname,name,middle)  {
entry =  surname  +  "  "  + name  +  "  "  + middle;
opt  = document.createElement('OPTION');
opt.value =  entry;
opt.text  = entry;
selector.options.add(opt,0);
}
```

Теперь записи из полей формы заносятся в раскрывающийся список (рис. 1.23).

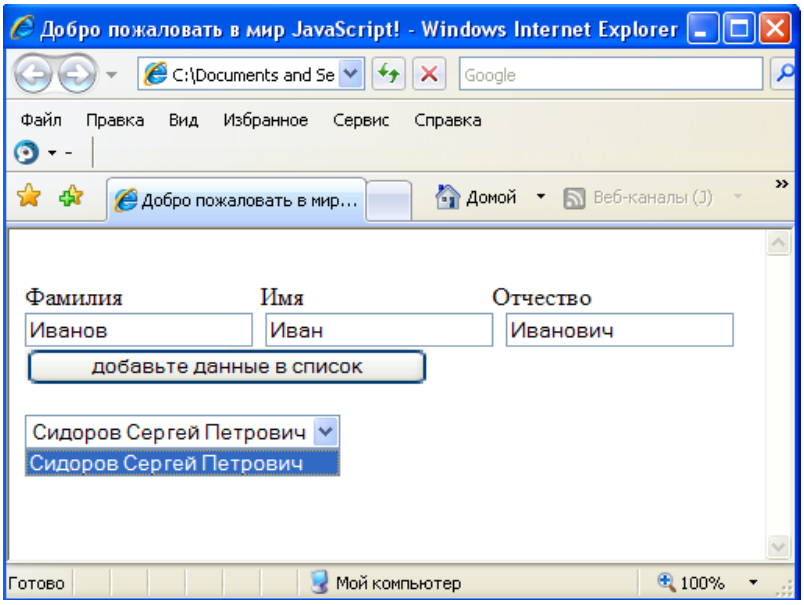

***Рис. 1.23**. Динамическое добавление пунктов в список*

В первой строке функции фамилия, имя и отчество, взятые из полей формы, объединяются в одну строку entry. Затем создается объект пункта списка opt, параметрам которого value и text присваивается полученная строка текста. Пункт списка добавляется в начало списка командой *имя_списка.* opt ions, add *(объект_пункта, индекс).*

Упражнение 7. Пример динамического изменения фона.

Листинг 1.21

Динамическое изменение фона Web-страницы

```
<HTML>
<HEAD>
<TITLE>Выбор фона</TITLE>
<SCRIPT>
function change__color (bg,fg) {
document .bgColor = bg;
document.fgColor = fg;
}
</SCRIPT>
<BODY BGCOLOR="ivory" ONFOCUS="document.bgColor='ivory' "
ONBLUR="document.bgColor='gray' ">
<H2>Выберите цвет фона</H2>
<FORM>
Щелкните на кнопке:
<BR><INPUT TYPE=button VALUE="Белый"
ONCLICK="change_color('white','black');">
<INPUT TYPE=button VALUE="Желтый"
ONCLICK="change_color('yellow','blue');">
<INPUT TYPE=button УАШЕ="Синий"
ONCLICK="change_color('blue','yellow');">
<INPUT TYPE=button VALUE="Черный"
ONCLICK="change_color('black','white');">
</BODY>
</HTML>
```

Страница содержит набор кнопок, изменяющих цвет фона и текста по умолчанию (рис. 1.24). Событиям ONCLICK кнопок присвоено выполнение функции change_color, в аргументах которой передаются соответствующие значения цветов.

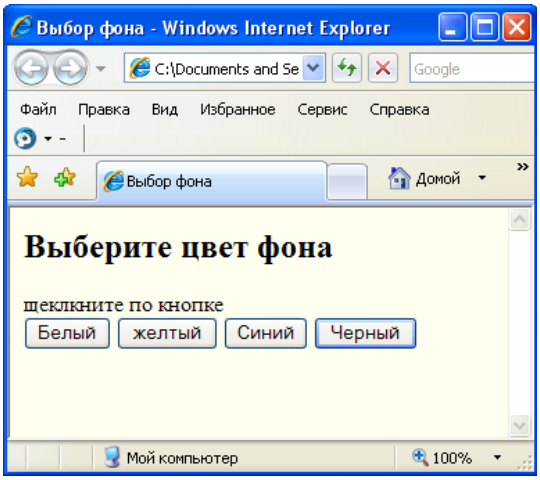

Рис. 1.24. *Щелчок на кнопке изменяет цвет фона и текста Web-страницы*

Упражнение 8. Работа с числовым массивом.

Листинг 1.22

Вывод на экран значений массива и суммы его положительных элементов

```
<BODY>
<SCRIPT>
var sum = 0;
var mas = [2, 3, 4, 5, 6, 4, 77, 32, 4];
for (var i = 0; i < mas.length; i ++) {
  document.write("Элемент номер " + i +' --- ' + mas[i] + "<br />");
 if(mas[i]>0) sum = sum +mas[i];
}
document.write("Сумма положительных элементов массива " +
sum + "<br />");
</SCRIPT>
</BODY>
```

Упражнение 9. Работа с массивом данных.

Листинг 1.23

Создание массива данных и управление им

```
<HTML>
<HEAD>
<TITLE>Массив</TITLE>
<SCRIPT>
function find_month ( ) {
month = prompt ("Введите месяц: ");
if (month) {
var i = NaN;
list = months.join ("$");
position = list.indexOf (month);
if (position > -1) {
var index = 0;
if (position) {
sub_list = list.substring (0,position);
sub_array = sub_list.split ("$");
index = sub_array.lendth - 1;
}
alert ("Месяц " + months[index] + " в массиве находтся под
индексом " + index);
}
else alert("Месяц " + month + " в массиве отсутствует");
}
}
</SCRIPT>
```

```
    </HEAD>
    <BODY>
    <SCRIPT>
    months = new Array("январь", "февраль", "март", "апрель",
"май", "июнь", "июль", "август", "сентябрь", "октябрь", "ноябрь");
    document.write("Исходный массив: ", months.toString(),
"<BR><BR>");
    months[11] = "декабрь";
    document.write("Новый массив: ", months.join("; "),
"<BR><BR>");
    spring_months = months.slice(2,5);
    autumn_months = months.slice(8,11);
    offseason = spring_months.concat(autumn_months);
    document.write("Весенние месяцы: ", spring_months.join("; "),
"<BR><BR>");
    document.write("Осенние месяцы: ", autumn_months.join("; "),
"<BR><BR>");
    document.write("Весенние и осенние месяцы: ", offseason.join("
"), "<BR><BR>");
    months.reverse();
    document.write("Инвертированный список: ", months.join("
"),"<BR><BR>");
    months.sort();
    document.write("месяцы в алфавитном порядке: ",
months.join("; "), "<BR><BR>");
    </SCRIPT>
    <FORM>
    <INPUT TYPE='button' VALUE='Найди месяц'
ONCLICK='find_month()'>
    </FORM>
    </BODY>
    </HTML>
```

В сценарии основного раздела Web-страницы создается массив month с названиями месяцев. Список месяцев выводится на страницу в виде строки текста с помощью методов document.write () и month.toString (). Затем мы добавляем в массив еще один месяц и выводим на экран новый список с помощью метода join. Далее демонстрируется использование методов slice и concat для разделения и объединения массивов и методов reverse и sort – для изменения порядка элементов в массиве. Результат выполнения сценария показан на рис. 1.25.

Сортировка массива изменила начальный порядок следования месяцев (см. рис. 1.25). Чтобы определить новый индекс месяца, на Web-страницу добавлена кнопка **Найди месяц.** Эта кнопка запускает на выполнение функцию find_month, определенную в сценарии в разделе заголовка. Функция открыва-

ет диалоговое окно для ввода названия месяца, после чего определяет индекс месяца в массиве и возвращает это значение в окне сообщения (см. рис. 1.25). В коде функции использовались методы строчных объектов, с которыми мы познакомимся ниже в этой главе.

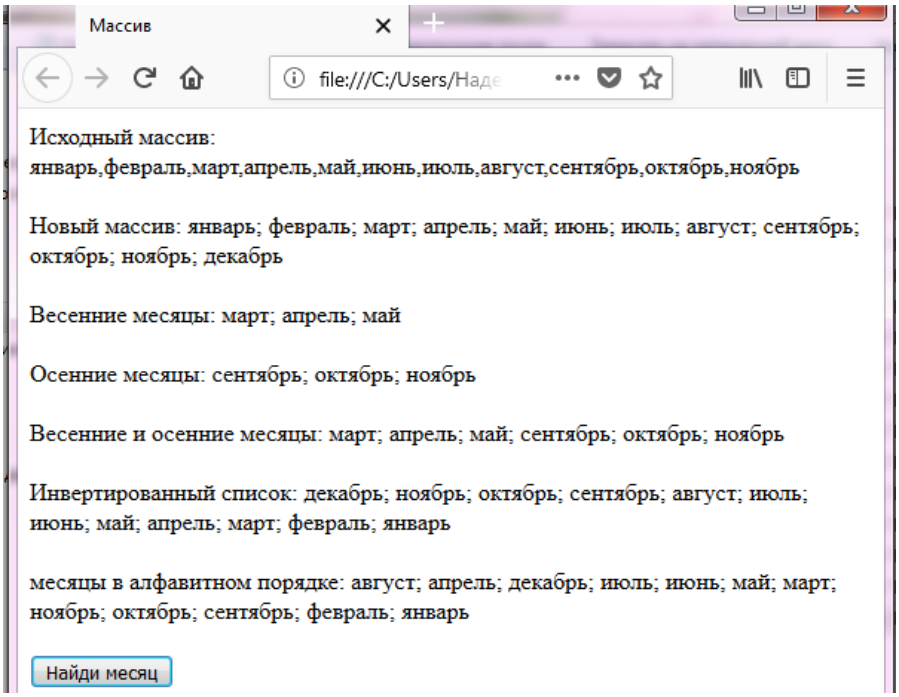

***Рис. 1.25**. Использование массивов в сценарии*

Упражнение 10. Работа с датой

Введите в текстовом редакторе код Web-страницы, показанный в листинге 1.24.

Листинг 1.24

Определение количества дней, оставшихся до Нового года

```
<HTML>
<HEAD>
<TITLE>Выбор фона</TITLE>
<SCRIPT>
function count() {
   now = new Date();
setdate = new Date("Jan 01 2018 00:00:00");

day = (now-setdate) / 1000 / 60 / 60 / 24;
   day = Math.round(day);
  document.write("До Нового года осталось " + "<b>" + day +
"</b>" + " дней!");
   }
```

```
    </SCRIPT>
    <FORM>
    Щелкните на кнопке:
    <BR><INPUT TYPE=button
ONCLICK="count()"VALUE="Результат">
    </BODY>
    </HTML>
```

В коде листинга задаются текущая дата и дата отсчета - 1 января текущего года. Вычисляется разность между этими датами, которая затем округляется в дни и выдается на экран по щелчку кнопки «Результат».

1.11. Задания для самостоятельного выполнения

Задание № 1. Последовательности

Сгенерировать последовательность с помощью датчика случайных чисел и обработать согласно варианту задания. **Массивы при выполнении этого задания не используются**.

1. Найти сумму элементов последовательности.
2. Найти минимальный элемент в последовательности.
3. Найти второй по величине элемент в последовательности.
4. Сколько раз в последовательности встречается заданное число?
5. Известно сопротивление каждого из элементов электрической цепи. Все элементы соединены параллельно. Определить общее сопротивление цепи.
6. Найти произведение элементов последовательности.
7. Найти сумму модулей элементов последовательности.
8. Сколько соответствующих элементов двух последовательностей с одинаковым количеством элементов совпадают?
9. Вычислить сумму квадратов элементов последовательности.
10. Определить среднее арифметическое элементов последовательности.
11. Определить среднее геометрическое элементов последовательности, содержащей положительные числа.
12. Найти произведение модулей элементов последовательности.
13. Определить, сколько раз встречается минимальный элемент в последовательности.
14. Определить, сколько раз встречается максимальный элемент в последовательности.

15. Сколько нулей в последовательности?

Задание № 2. Задания с датой и временем

Примечание. Текущие дата и время определяются системным временем. При тестировании скрипта дата и время могут быть изменены на любые допустимые.

1. Написать скрипт, определяющий, через сколько дней наступит воскресенье.

2. Написать скрипт, определяющий, сколько дней прошло с Нового года.

3. Составить скрипт, определяющий сколько часов и минут прошло от начала суток.

4. Определить, сколько дней прошло с начала текущего месяца.

5. Определить, сколько часов и минут прошло с начала текущего месяца.

6. Через сколько часов (по гринвичскому времени) наступит Новый год?

7. Вывести полную информацию о текущей дате и времени. Например, "14 мая 2002 года, вторник, 2:53:44pm".

8. Определить, сколько недель осталось до 1 сентября.

9. Определить, сколько недель прошло с 1 сентября.

10. Определить, сколько дней сталось до Вашего дня рождения.

11. Сколько часов осталось до начала лета?

12. Сколько суток осталось до дней весеннего и осеннего равноденствий (22 марта и 22 сентября)?

13. Сколько дней прошло со дня Вашего рождения?

14. Сколько дней осталось до ближайшей пятницы, выпадающей на 13-е число?

15. Вывести названия месяцев текущего года, где пятница выпадает на 13-е число.

Задание № 3. Обработка событий

Напишите функции-обработчики для следующих событий:

onClick

1. При нажатии на кнопку button_1 пользователь получает одно из трех сообщений: "Кликни-ка еще разик", "Эта кнопка — самая замечательная кнопка в мире", "Ну чего раскликался тут?!".

2. При нажатии на кнопку button_1 должно происходить закрытие окна документа.

3. При нажатии на ссылку вывести пользователю сообщение: "И куда это вы собрались?".

4. При нажатии на картинку image_1 изображение должно случайным образом меняться на одно из пяти.

onMouseOver

5. При наведении курсора мыши на картинку изображение должно меняться на другое.

6. При наведении курсора мыши на кнопку должно выдаваться некоторое сообщение.

7. При наведении курсора мыши на ссылку в строке состояния (window.status) должно выводиться название соответствующего ресурса.

8. При наведении курсора мыши на какой-либо элемент страницы меняется цвет фона документа (document.bgColor).

onMouseOut

9. При наведении курсора мыши на картинку изображение должно меняться на другое, а когда курсор покинет область картинки, должно восстанавливаться прежнее изображение

onChange

10. При изменении значения в текстовом поле text_1 должно выводиться: «Быстро сделай все, как было!»

onSelect

11. При выделении какого-либо элемента страницы в строке состояния должно появиться сообщение: «Выделять-то можно, копировать — нельзя!»

onLoad

12. При завершении загрузки документа должно выводиться приветствие пользователю

onCopy и onCut

13. При попытке пользователя скопировать что-либо со страницы должно выводиться сообщение: «Информация на данном сайте строго конфиденциальна, разглашению и копированию не подлежит» (для запрещения копирования функция–обработчик должна возвращать значение false)

onDblclick

14. При двойном щелчке мыши по картинке она должна увеличиваться в размерах

onHelp

15. Написать функцию, запрещающую воспользоваться справкой, выдавая соответствующее сообщение.

onResize

16. Вывести пользователю предупреждение об ответственности за изменение размеров окна браузера.

onSubmit

17. Написать функцию, которая при нажатии пользователя на кнопку submit выдавала сообщение: «Благодарим, что написали нам»

onUnload

18. Написать функцию, которая выдавала бы сообщение «Вы нас уже покидаете?», когда пользователь уходит со страницы.

Задание № 4. Работа с массивами

Задание 1. Дан массив mas. Выведите его на страницу HTML в формате «индекс элемента --- значение» (через три дефиса). Каждый элемент с новой строки.

Задание 2. Дан массив mas. Выведите его на страницу HTML в формате «индекс элемента — значение» (через дефис). Каждый элемент с новой строки. Выводить нужно те элементы, значение которых больше пяти (5).

Задание 3. Дан массив mas. Выведите на страницу HTML максимальный элемент массива, указав его индекс.

Задание 4. Дан массив mas. Выведите на страницу HTML среднее арифметическое всех элементов массива.

Задание 5. Дан массив mas. Выведите на страницу HTML четные элементы массива, указав их индексы.

Задание 6. Дан массив mas. Выведите на страницу HTML нулевые элементы массива, указав их индексы.

Задание 7. Дан массив mas. Выведите на страницу HTML нечетные элементы массива, указав их индексы.

Задание 8. Дан массив mas. Выведите на страницу HTML те элементы массива, которые оканчиваются на 4. При выводе укажите их индексы.

Задание 9. Дан массив mas. Выведите на страницу HTML те элементы массива, которые оканчиваются на 7. При выводе укажите их индексы.

Задание 10. Дан массив mas. Выведите на страницу HTML положительные элементы массива, указав их индексы.

Задание 11. Дан массив mas. Выведите на страницу HTML минимальный элемент массива, указав его индекс.

Задание 12. Дан массив mas. Выведите на страницу HTML отрицательные элементы массива, указав их индексы.

Задание 13. Дан массив mas. Выведите на страницу HTML те элементы массива, которые оканчиваются на 9. При выводе укажите их индексы.

Задание 14. Дан массив mas. Выведите на страницу HTML те элементы массива, которые кратны 5. При выводе укажите их индексы.

Задание 15. Дан массив mas. Выведите на страницу HTML индексы тех элементов массива, которые равны максимальному элементу массива.

Задание № 5. Работа с формой

Разработайте форму, заполняемую пользователем Internet, на произвольную тематику. При этом должны быть задействованы все виды элементов, которые могут присутствовать на форме.

Разработайте скрипт, проверяющий правильность заполнения полей формы (не пустые ли они, соответствуют ли типы и диапазоны данных и др.) При неправильном заполнении сообщить, какие поля надо исправить. Если всё заполнено, верно, сообщить, что данные отправлены.

Глава 2. Общий обзор языка программирования PHP

PHP является наиболее распространенным языком веб-программирования. Простота языка позволяет быстро и легко создавать сайты и порталы различной сложности. Отметим следующие преимущества PHP:

- версии пакетов разработки на PHP есть для всех распространенных операционных систем (Windows, MacOS, Linux);
- PHP может работать в связке с различными веб-серверами: Apache, Nginx и др.;
- простота и легкость программирования на PHP;
- PHP поддерживает работу с множеством систем баз данных (MySQL, MSSQL, Oracle, Postgre, MongoDB и др.).

2.1. Основы PHP. Основы синтаксиса

Программа или скрипт на PHP, как правило, находится в файле расширением *.php*. Хотя разработчики могут также вставлять код php и в файлы с расширениями *.html/.htm*.

Когда пользователь обращается к скрипту в адресной строке браузера, набирая, например, *http://localhost:8080/display.php*, то веб-сервер передает его интерпретатору PHP. Затем интерпретатор обрабатывает код и генерирует на его основе html-разметку. И затем сгенерированный html-код отправляется пользователю.

Документ PHP может содержать как разметку html, так и код на языке php. Для перехода от разметки html к коду php используются теги `<php` и `>`, между которыми идет код php. Данные теги служат указанием интерпретатору, что их содержимое надо интерпретировать как код php, а не разметку html.

Также можно использовать краткую версию тегов: `<` и `>`. Для этого в файле *php.ini* надо изменить значение `short_open_tag = Off` на `short_open_tag = On`

Рассмотрим простейший скрипт на php:

```
<html>
<head>
<title>Веб-сайт</title>
</head>
<body>
<?php
echo "<p>Привет мир!</p>";
echo "2 + 2 = " . (2+2);
?>
</body>
</html>
```

После обработки файла интерпретатор сформирует следующую разметку:

```
<html>
<head>
<title>Веб-сайт</title>
</head>
<body>
<p>Привет мир!</p>
2 + 2 = 4
</body>
</html>
```

Здесь использованы две инструкции `echo "<p>Привет мир!</p>"` и `echo "2 + 2 = " . (2+2)`, который выводят определенное значение на страницу. Каждая отдельная инструкция в PHP завершается точкой с запятой.

Комментарии

При создании веб-сайта мы можем использовать комментарии. Например, мы можем прокомментировать какое-либо действие, чтобы впоследствии иметь представление, что этот код делает:

```
<?php
echo "<p>Привет мир!</p>"; // вывод сообщения
// echo "2 + 2 = " . (2+2);
?>
```

Знак `//` предваряет однострочный комментарий, и все что идет после этого знака на одной строке, будет считаться комментарием и не будет выполняться интерпретатором. При обработке интерпретатор просто будет пропускать комментарии.

Если нам надо закомментировать несколько строк, то мы можем использовать многострочный комментарий `/* текст комментария*/`:

```
<?php
echo "<p>Привет мир!</p>"; // вывод сообщения
/*
многострочный комментарий
вывод результата арифметического выражения
echo "2 + 2 = " . (2+2);
*/
?>
```

Все строки внутри комментария также не будут обрабатываться интерпретатором.

Переменные

Переменные хранят отдельные значения, которые можно использовать в выражениях на PHP. Для обозначения переменных используется знак доллара `$`. Например,

```
<?php
$a = 10;
echo $a;
?>
```

Здесь определена переменная, которая будет хранить число 10. Присвоение значения происходит с помощью знака равенства =.

Можно присваивать значение другой переменной:

```
$a = 10;
$b=$a;
echo $b;
```

PHP является *регистрозависимым* языком, а значит, переменные `$counter` и `$Counter` будут представлять две разные переменные.

Также при наименовании переменных нам надо учитывать следующие правила:

- Имена переменных должны начинаться с алфавитного символа или с подчеркивания.
- Имена переменных могут содержать только символы: a–z, A–Z, 0–9, и знак подчеркивания.
- Имена переменных не должны включать в себя пробелы.

Проверка существования переменной. Оператор isset.

Если переменная объявлена, но ей изначально не присвоено никакого значения (т.е. она не инициализирована), то использовать ее не получится. Например:

```
<?php
$a;
echo $a;
?>
```

При попытке вывести значение переменной мы получим диагностическое сообщение о том, что переменная не определена: *Notice: Undefined variable: a in C:\localhost\echo.php on line 3*.

Оператор **isset()** позволяет определить, инициализирована ли переменная или нет. Если переменная определена, то `isset()` возвращает значение `true`. Если переменная не определена, то `isset()` возвращает `false`. Например:

```
<?php
$a;
if(isset($a))
   echo $a;
else
   echo "переменная a не определена";
?>
```

Для проверки переменной использовалась конструкция **`if...else`**, которая определяет истинность выражения. И если выражение истинно, тогда выполняется выражение после блока **if**. Если же выражение ложно (то есть равно false), выполняется выражение после блока **`else.`**

В примере переменная не инициализирована, поэтому оператор `isset($a)` будет возвращать значение `false`, и, следовательно, будет срабатывать блок `else`. Если бы переменной присвоили некоторое начальное значение, например, `$a=20`, то оператор isset возвратил бы значение `true`, и браузер вывел бы ее значение.

С помощью оператора **unset()** мы можем уничтожить переменную:

```
<?php
$a=20;
echo $a; // 20
unset($a);
echo $a; // ошибка, переменная не определена
?>
```

2.2. Типы данных

PHP является языком с динамической типизацией, т.е. тип данных переменной выводится во время выполнения, и в PHP нет необходимости указывать перед переменной тип данных.

PHP поддерживает восемь простых типов данных:

boolean (логический тип)	string (строки)	resource (ресурсы)
integer (целые числа)	array (массивы)	NULL
double (дробные числа)	object (объекты)	

Integer (целочисленный тип) – целое число со знаком размером в 32 бита (от -2 147 483 648 до 2 147 483 647). Например:

```
$int = -100;
echo $int;
```

Кроме десятичных целых чисел PHP обладает возможностью использовать также двоичные, восьмеричные и шестнадцатеричные числа. Шаблоны чисел для других систем:

шестнадцатеричные : 0[xX][0-9a-fA-F]

восьмеричные : 0[0-7]

двоичные : 0b[01]

Например:

```
<?php
// Все числа в десятичной системе имеют значение 28
$int_10 = 28; // десятичное число
$int_2 = 0b11100; // двоичное число
$int_8 = 034; // восьмеричное число
$int_16 = 0x1C; // шестнадцатеричное число
echo "int_10 = $int_10 <br>";
echo "int_2 = $int_2 <br>";
echo "int_8 = $int_8 <br>";
echo "int_16 = $int_16";
?>
```

Double (числа с плавающей точкой) – размер числа с плавающей точкой зависит от платформы. Максимально возможное значение, как правило, составляет ~1.8e308 с точностью около 14 десятичных цифр. Например:

```
<?php
$a1 = 1.5;
$a2 = 1.3e4; // 1.3 * 10^4
$a3 = 6E-8; // 0.00000006
echo $a1 . " | " . $a2 . " | " . $a3;
?>
```

Тип boolean (логический тип)

Переменные логического типа могут принимать два значения: `true` и `false`. Чаще всего логические значения используются в условных конструкциях:

```
<?php
$foo = true;
$a=10;
$b=5;
echo "foo = true <br>";
if($foo)
  echo $a+$b;
else
  echo $a-$b;
$foo = false;
echo "<br> foo = false <br>";
if($foo)
  echo $a+$b;
else
  echo $a-$b;
?>
```

Специальное значение NULL

Значение `NULL` указывает, что значение переменной не определено. Использование данного значения полезно в тех случаях, когда мы хотим указать, что переменная не имеет значения. Например, если мы просто определим переменную без ее инициализации, и затем попробуем ее использовать, то нам интерпретатор выдаст диагностическое сообщение, что переменная не установлена:

```
<?php
$a;
echo $a;
?>
```

Использование значения NULL поможет избежать данной ситуации. Кроме того, мы сможем проверять наличие значения и в зависимости от результатов проверки производить те или иные действия:

```
<?php
$a=NULL;
if($a)
   echo "Переменная a определена";
else
   echo "Переменная a не определена";
?>
```

Константа NULL не чувствительна к регистру, поэтому мы можем написать и так: `$a=null;`

Тип string (строки)

Для работы с текстом можно применять строки. Строки бывают двух типов: в двойных кавычках и одинарных. От типа кавычек зависит обработка строк интерпретатором. Так, переменные в двойных кавычках заменяются значениями, а переменные в одинарных кавычках остаются неизменными.

```
<?php
$a=10;
$b=5;
$result = "$a+$b <br>";
echo $result;
$result = '$a+$b';
echo $result;
?>
```

В этом случае мы получим следующий вывод:

```
10+5
$a+$b
```

Тип resource (ресурсы)

Ресурс представляет специальную переменную, которая содержит ссылку на внешний ресурс. В качестве внешнего ресурса могут использоваться, например, файлы или подключения к базам данных. Ресурсы создаются и используются специальными функциями.

Тип array (ассоциативные массивы)

Ассоциативный массив определяет набор элементов, каждый из которых представляет пару `ключ=>значение`. Создадим массив из 4 элементов:

```
<?php
$phones = array('iPhone', 'Samsung Galaxy S III', 'Nokia N9', 'Samsung ACE II');
echo $phones[1];
?>
```

Массив создается с помощью конструкции `array()`, в которой определяются элементы. Далее выводим второй элемент массива. Поскольку отсчет элементов в массиве начинается с нуля, то чтобы обратиться ко 2-му элементу, нам надо использовать выражение `$phones[1]`

Так как в массиве только четыре элемента, мы не можем использовать в качестве ключа число, большее чем 3, например, `$phones[4]`.

Константы

Константы, как и переменные, хранят определенное значение, только в отличие от переменных, значение констант может быть установлено только один раз, и далее мы уже не можем его изменить. Например, определим числовую константу:

```
<?php
define("NUMBER", 22);
echo NUMBER;
$num = NUMBER;
echo $num;
?>
```

Для определения константы используется оператор **define**, который имеет следующую форму: define(string $name, string $value, bool $case_sen=false). Параметр $name передает название константы, а параметр $value - ее значение. 3-й необязательный параметр принимает логическое значение true или false. Если значение равно false, то при использовании константы будет учитываться ее регистр, если true - регистр не учитывается. В нашем случае 3-й параметр не использован, поэтому он по умолчанию равен false.

После определения константы мы можем ее использовать так же, как и обычную переменную. Единственное исключение - мы не сможем изменить ее значение. Другое отличие от переменной - не надо использовать знак $.

Предопределенные константы

Кроме создаваемых программистом констант в PHP имеется еще несколько встроенных констант:

__FILE__: хранит полный путь и имя текущего файла.

__LINE__: хранит текущий номер строки, которую обрабатывает интерпретатор.

__DIR__: хранит каталог текущего файла.

__FUNCTION__: название обрабатываемой функции.

__CLASS__: название текущего класса.

__METHOD__: название обрабатываемого метода.

__NAMESPACE__: название текущего пространства имен.

Например, выведем текущую выполняемую строку и название файла:

```
<?php
echo "Строка " . __LINE__ . " в файле " . __FILE__;
?>
```

Проверка существования константы

Чтобы проверить, определены ли константы, мы можем использовать функцию **bool defined(string $name)**. Если константа `$name` определена, то функция будет возвращать значение `true`.

Получение и установка типа переменной

С помощью специальных функций мы можем определить тип переменной:

is_integer($a): возвращает значение `TRUE`, если переменная $a хранит целое число;

is_string($a): возвращает значение `TRUE`, если переменная $a хранит строку;

is_double($a): возвращает значение `TRUE`, если переменная $a хранит действительное число;

is_numeric($a): возвращает значение `TRUE`, если переменная $a представляет целое или действительное число или является строковым представлением числа. Например:

```
$a = 10;
$b = "10";
echo is_numeric($a);
echo "<br>";
echo is_numeric($b);
```

Оба выражения `is_numeric()` возвратят TRUE, так как переменная $a представляет число, а переменная $b является строковым представлением числа.

is_bool($a): возвращает значение `TRUE`, если переменная $a хранит значение TRUE или FALSE.

is_scalar($a): возвращает значение `TRUE`, если переменная $a представляет один из простых типов: строку, целое число, действительное число, логическое значение.

is_null($a): возвращает значение `TRUE`, если переменная $a хранит значение `NULL`.

is_array($a): возвращает значение `TRUE`, если переменная $a является массивом.

is_object($a): возвращает значение `TRUE`, если переменная $a содержит ссылку на объект.

gettype($a): возвращает тип переменной $a, например, `integer` (целое число), `double` (действительное число), `string` (строка), `boolean` (логическое значение), `NULL`, `array` (массив), `object` (объект) или `unknown type`. Например:

```
<?php
$a = 10;
$b = "10";
echo gettype($a); // integer
echo "<br>";
echo gettype($b);  // string
?>
```

Установка типа. Функция settype()

С помощью функции **settype()** можно установить для переменной определенный тип. Она принимает два параметра: `settype("Переменная", "Тип")`. В качестве первого параметра используется переменная, тип которой надо установить, а в качестве второго - строковое описание типа, которое возвращается функцией gettype().

Если удалось установить тип, то функция возвращает TRUE, если нет - то значение FALSE.

Например, установим для переменной целочисленный тип:

```
<?php
$a = 10.7;
settype($a, "integer");
echo $a; // 10
?>
```

Поскольку переменная $a представляет действительное число 10.7, то его вполне можно преобразовать в целое число через отсечение дробной части. Поэтому в данном случае функция settype() возвратит TRUE.

2.3. Операции в PHP

В PHP мы можем использовать различные операторы: арифметические, логические и т.д. Основные типы операций представлены в табл.2.1.

Таблица 2.1

Арифметические операции	
+ (операция сложения)	$a + 5
- (операция вычитания)	$a - 5
* (умножение)	$a * 5
/ (деление)	$a / 5
% (получение остатка от деления)	$a=12; echo $a % 5; // равно 2
++ (инкремент/ увеличение значения на единицу)	++$a; $a++;
-- (декремент/ уменьшение значения на единицу)	--$a; $a--;

Продолжение таблицы 2.1

Операции присваивания	
=	Приравнивает переменной определенное значение: $a = 5
+=	Сложение с последующим присвоением результата. $a=12; $a += 5; echo $a; // равно 17
-=	Вычитание с последующим присвоением результата.
*=	Умножение с последующим присвоением результата.
/=	Деление с последующим присвоением результата.
.=	Объединение строк с присвоением результата. Применяется к 2 строкам. Если же переменные хранят не строки, а, к примеру, числа, то их значения преобразуются в строки и затем проводится операция: $a=12; $a .= 5; echo $a; // равно 125
%=	Получение остатка от деления с последующим присвоением результата: $a=12; $a %= 5; echo $a; // равно 2
Операции сравнения	
= =	Оператор равенства сравнивает 2 значения, и если они равны, возвращает true, иначе false: $a = = 5
= = =	Оператор тождественности также сравнивает два значения, и если они равны, возвращает true, иначе возвращает false: $a = = = 5
!=	Сравнивает два значения, и если они **не** равны, возвращает true, иначе возвращает false: $a != 5
!= =	Сравнивает два значения, и если они **не** равны, возвращает true, иначе возвращает false: $a !== 5

Окончание таблицы 2.1

>	Сравнивает два значения, и если первое больше второго, то возвращает true, иначе возвращает false: $a > 5
<	Сравнивает два значения, и если первое меньше второго, то возвращает true, иначе возвращает false
>=	Сравнивает два значения, и если первое больше или равно второму, то возвращает true, иначе возвращает false
<=	Сравнивает два значения, и если первое меньше или равно второму, то возвращает true, иначе возвращает false
Логические операции	
&&	Возвращает true, если обе операции сравнения возвращают true, иначе возвращает false: $a == 5 && $b = 6
and	Аналогично операции &&: $a == 5 and $b > 6
||	Возвращает true, если хотя бы одна операция сравнения возвращает true, иначе возвращает false: $a == 5 || $b = 6
or	Аналогично операции ||: $a < 5 or $b > 6
!	Возвращает true, если операция сравнения возвращает false: !($a >= 5)

xor	Возвращает true, если только одно из значений равно true. Если оба равны true или ни одно из них не равно true, возвращает false. Например:
	$a=12;
	$b=6;
	if($a xor $b)
	  echo 'true';
	else
	  echo 'false';

Оператор равенства и тождественности

Оба оператора сравнивают два выражения и возвращают true, если выражения равны. Но между ними есть различия. Если в операции равенства принимают два значения разных типов, то они приводятся к одному – тому, который интерпретатор найдет оптимальным. Например:

```
<?php
$a = "22a";
$b = 22;
if($a==$b)
   echo "равны";
else
   echo "не равны";
?>
```

Очевидно, что переменные хранят разные значения разных типов. Но при сравнении они будут приводится к одному типу – числовому. И переменная $a будет приведена к числу 22. И в итоге обе переменных окажутся равны.

Чтобы избежать подобных ситуаций, используется операция эквивалентности, которая учитывает не только значение, но и тип переменной:

```
$a = "22a";
$b = 22;
if($a===$b)
   echo "равны";
else
   echo "не равны";
```

Теперь переменные будут не равны.

Аналогично работают операторы неравенства != и !==.

Объединение строк

Для объединения строк используется оператор "точка". Например, соединим несколько строк:

```
$a="Привет, ";
$b=" мир";
echo $a . $b . "!";
```

Если переменные представляют не строки, а другие типы, например, числа, то их значения преобразуются в строки, и затем также происходит операция объединения строк.

Условные конструкции

Условные конструкции позволяют направлять работу программы в зависимости от условия по одному из возможных путей.

Конструкция if..else

Конструкция `if (условие)` проверяет истинность некоторого условия, и если оно окажется истинным, то выполняется блок выражений, стоящих после if. Если же условие ложно, то есть равно false, тогда блок if не выполняется. Например:

```
<?php
$a = 4;
$b = 2;
if($a>0)
{
   $result= $a * $b;
   echo "результат равен: $result <br>";
}
echo "конец выполнения программы";
?>
```

Блок выражений ограничивается фигурными скобками. И так как в данном случае условие истинно (то есть равно true): значение переменной $a больше 0, то блок инструкций в фигурных скобках также будет выполняться. Если бы значение $a было бы меньше 0, то блок if не выполнялся.

Если блок if содержит всего одну инструкцию, то можно опустить фигурные скобки:

```
<?php
$a = 4;
$b = 2;
if($a>0)
   echo $a * $b;
echo "<br>конец выполнения программы";
?>
```

Блок else содержит инструкции, которые выполняются, если условие после if ложно, то есть равно false:

```
<?php
$a = 4;
$b = 2;
if($a>0)
{
```

```
    echo $a * $b;
}
else
{
    echo $a / $b;
}
echo "<br>конец выполнения программы";
?>
```

Если $a больше 0, то выполняется блок if, если нет, то выполняется блок else.

elseif

Конструкция elseif вводит дополнительные условия в программу:

```
$a = 5;
$b = 2;
if($a<0)
{
    echo $a * $b;
}
elseif($a==0)
{
    echo $a + $b;
}
elseif($a==5)
{
    echo $a - $b;
}
else
{
    echo $a / $b;
}
```

Можно добавить множество блоков `elseif`. И если ни одно из условий в if или elseif не выполняется, тогда срабатывает блок else.

Конструкция switch..case

Конструкция `switch..case` является альтернативой использованию конструкции `if..elseif..else`. Например:

```
$a = 1;
if($a==1)     echo "сложение";
elseif($a==2) echo "вычитание";
elseif($a==3) echo "умножение";
elseif($a==4) echo "деление";
```

Будет эквивалентно:

```
$a = 1;
switch($a)
{
    case 1:
        echo "сложение";
        break;
    case 2:
        echo "вычитание";
        break;
    case 3:
        echo "умножение";
        break;
    case 4:
        echo "деление";
        break;
}
```

После ключевого слова switch в скобках идет сравниваемое выражение. Значение этого выражения последовательно сравнивается со значениями, помещенными после оператора case. И если совпадение будет найдено, то будет выполняться определенный блок case.

В конце блока case ставится оператор break, чтобы избежать выполнения других блоков.

Если мы хотим также обработать ситуацию, когда совпадения не будет найдено, то можно добавить блок default:

```
$a = 1;
switch($a)
{
    case 1:
        echo "сложение";
        break;
    case 2:
        echo "вычитание";
        break;
    default:
        echo "действие по умолчанию";
        break;
}
```

Тернарная операция

Тернарная операция состоит из трех операндов и имеет следующее определение: `[первый операнд - условие] ? [второй операнд] : [третий операнд]`. В зависимости от условия тернарная операция возвращает второй или третий операнд: если условие равно `true`, то возвращается второй операнд; если условие равно `false`, то третий. Например:

```
$a = 1;
$b = 2;
$z = $a < $b ? $a + $b : $a - $b;
echo $z;
```

Если значение переменной $a меньше $b и условие истинно, то переменная $z будет равняться `$a + $b`. Иначе значение $z будет равняться`$a - $b`

2.4. Циклы

Для совершения повторяемых действий в PHP, как и в других языках программирования, используются циклы. В PHP имеются следующие виды циклов: for, while, do..while

Цикл for

Цикл for имеет следующее формальное определение:

```
for ([инициализация счетчика]; [условие]; [изменение счетчика])
{
    // действия
}
```

Рассмотрим стандартный цикл for:

```
<?php
for ($i = 1; $i < 10; $i++)
{
    echo "Квадрат числа $i равен " . $i * $i . "<br/>";
}
?>
```

Первая часть объявления цикла - `$i = 1` - создает и инициализирует счетчик – переменную $i. Перед выполнением цикла его значение будет равно 1. По сути это то же самое, что и объявление переменной.

Вторая часть – условие, при котором будет выполняться цикл. В данном случае цикл будет выполняться, пока $i не достигнет 10.

Третья часть – приращение счетчика на единицу. Нам необязательно увеличивать на единицу. Можно уменьшать: $i--.

В итоге блок цикла сработает 9 раз, пока значение $i не станет равным 10.

Цикл while

Цикл `while` проверяет истинность некоторого условия, и если условие истинно, то выполняется блок выражений цикла:

```
<?php
$counter = 1;
while($counter<10)
{
   echo $counter * $counter . "<br />";
   $counter++;
}
?>
```

Если в блоке while всего одна инструкция, то фигурные скобки блока можно опустить.

Цикл do..while

Цикл `do..while` похож на цикл `while`, только теперь выполняется блок цикла, и только потом выполняется проверка условия, то есть даже если условие ложно, то блок цикла выполнится как минимум один раз:

```
<?php
$counter = 1;
do
{
   echo $counter * $counter . "<br />";
   $counter++;
}
while($counter<10)
?>
```

Операторы continue и break

Иногда возникает ситуация, когда требуется выйти из цикла, не дожидаясь его завершения. В этом случае мы можем воспользоваться оператором break:

```
<?php
for ($i = 1; $i < 10; $i++)
{
    $result = $i * $i;
    if($result>80)
    {
        break;
    }
    echo "Квадрат числа $i равен $result <br/>";
}
?>
```

И если вдруг результат операции окажется > 80, то происходит выход из цикла.

Для управления циклами также применяется оператор **continue**. Он осуществляет переход к следующей итерации цикла:

```
<?php
for ($i = 1; $i < 10; $i++)
{
    if($i==5)
    {
        continue;
    }
    echo "Квадрат числа $i равен " . $i * $i . "<br/>";
}
?>
```

При выполнении программы, когда значение $i станет равным 5, произойдет переход к следующей итерации, а все остальные выражения, которые следуют после оператора `continue`, выполняться не будут.

2.5. Функции

Функции представляют собой набор инструкций, выполняющих определенное действие.

Синтаксис определения функции:

```
function имя_функции([параметр [, ...]])
{
    // Инструкции
}
```

Определение функции начинается с ключевого слова **function**, за которым следует имя функции. Имя функции должно начинаться с алфавитного символа или подчеркивания, за которыми может следовать любое количество алфавитно-цифровых символов или символов подчеркивания.

После имени функции в скобках идет перечисление параметров. Даже если параметров у функции нет, то просто идут пустые скобки. Затем в фигурных скобках идет тело функции, содержащее набор инструкций.

Определим простейшую функцию:

```
function display()
{
    echo "вызов функции display()";
}
```

Данная функция не имеет параметров, и все, что она делает - это выводит на страницу некоторое сообщение.

Чтобы функция сработала, ее надо вызвать. Теперь вызовем функцию:

```
<?php
display();
function display()
{
echo "вызов функции display()";
}
?>
```

Возвращение значения и оператор return

Функция может возвращать некоторое значение - число, строку и т.д., то есть некоторый результат. Для возвращения значения в функции применяется

оператор **return**, после которого указывается возвращаемое значение. Например:

```
<?php

$a = get();
echo "Сумма квадратов от 1 до 9 равна $a";

function get()
{
    $result = 0; // возвращаемое значение
    for($i = 1; $i<10; $i++)
    {
        $result+= $i * $i;
    }
    return $result;
}
?>
```

Функция `get()` возвращает число, представляющее сумму квадратов от 1 до 9. Это число хранится в переменной `$result`. Благодаря оператору return мы можем присвоить значение, возвращаемое функцией get, какой-нибудь переменной: `$a = get();`.

Использование параметров

Создадим функцию с параметрами:

```
<?php

$a = get(1, 10);
echo "Сумма квадратов от 1 до 9 равна $a";

function get($lowlimit, $highlimit)
{
    $result = 0; // возвращаемое значение
    for($i = $lowlimit; $i < $highlimit; $i++)
    {
        $result+= $i * $i;
    }
    return $result;
}
?>
```

Так как теперь функция $get использует параметры, то мы должны передать при вызове этой функции на место параметров некоторые значения. Если при вызове мы укажем значения не для всех параметров, то это будет ошибка, например: `$a = get(1);`.

Но мы можем использовать значения по умолчанию для параметров. Например:

```
<?php
function get($lowlimit, $highlimit=10)
{
    $result = 0; // возвращаемое значение
    for($i = $lowlimit; $i < $highlimit; $i++)
    {
        $result+= $i * $i;
    }
    return $result;
}

$a = get(1);
echo "Сумма квадратов равна $a";
?>
```

В этом случае, если мы не укажем значение для второго параметра, то по умолчанию он будет равен 10.

Передача по ссылке

В примере выше мы передавали параметры **по значению**. Но в PHP есть и другая форма передачи параметров – **по ссылке**. Рассмотрим два этих способа передачи параметров и сравним. Стандартная передача параметра по значению:

```
<?php
$number = 10;
get($number);
echo "<br /> \$number равно: $number";

function get($a)
{
    $a*=$a;
    echo "Квадрат равен: $a";
}
?>
```

После вызова функции `get()` значение переменной `$number` не изменится, так как в эту функцию в качестве параметра мы передаем значение переменной.

Теперь рассмотрим передачу параметра по ссылке:

```
<?php
$number = 10;
get($number);
echo "<br /> \$number равно: $number";
function get(&$a)
{
   $a*=$a;
   echo "Квадрат равен: $a";
}
?>
```

При передаче по ссылке перед параметром ставится знак амперсанда: `function get(&$a)`. Теперь интерпретатор будет передавать не значение переменной, а ссылку на эту переменную в памяти, в итоге, переменная `$number` после передачи на место параметра `&$a` также будет изменяться.

Область видимости переменной

При использовании переменных и функций следует учитывать области видимости переменных. Область видимости задает область действия, доступности данной переменной. Существуют локальные, статические и глобальные переменные.

Локальные переменные создаются внутри функции. К таким переменным можно обратиться только изнутри данной функции. Как правило, локальные переменные хранят какие-то промежуточные результаты вычислений, как в примере ниже.

```
<?php
function get($lowlimit, $highlimit)
{
   $result = 0; // возвращаемое значение
   for($i = $lowlimit; $i < $highlimit; $i++)
   {
      $result+= $i * $i;
   }
   return $result;
}
$a = $result; // так нельзя написать, так как $result - локальная переменная
echo "Сумма квадратов от 1 до 9 равна $a";
?>
```

В данном случае в функции `get()` определена локальная переменная `$result`. И из общего контекста мы не можем к ней обратиться, то есть написать `$a = $result;` нельзя, так как область действия переменной `$result` ограничена функцией `get()`. Вне этой функции переменной `$result` не существует.

То же самое относится и к параметрам функции: вне функции параметры $lowlimit и $highlimit также не существуют.

Статические переменные похожи на локальные. Они отличаются тем, что после завершения работы функции их значение сохраняется. При каждом новом вызове функция использует ранее сохраненное значение. Как правило, статические переменные служат для создания различных счетчиков, например:

```
<?php
function getCounter()
{
    static $counter = 0;
    $counter++;
    echo $counter;
}
getCounter(); // counter=1
getCounter(); // counter=2
getCounter(); // counter=3
?>
```

Чтобы указать, что переменная будет статической, к ней добавляется ключевое слово **static**. При трех последовательных вызовах функции `getCounter()` переменная $counter будет увеличиваться на единицу.

Если бы переменная $counter была обычной нестатической, то при каждом вызове функция getCounter() выводила бы 1.

Иногда требуется, чтобы переменная была доступна везде, глобально. Подобные переменные могут хранить какие-то общие для всей программы данные. Для определения глобальных переменных используется ключевое слово **global**:1

```
<?php
function getGlobal()
{
    global $gvar;
    $gvar = 20;
    echo "$gvar <br />";
}
getGlobal();
echo $gvar;
?>
```

После вызова функции `getGlobal()` к переменной `$gvar` можно будет обратиться из любой части программы.

2.6. Подключение внешних файлов

Если программа небольшая, текста программы немного, то все операции можно определить и в одном файле. Однако, как правило, программы состоят из множества инструкций. При определении всех этих инструкций в одном файле код может выглядеть слишком громоздким. Поэтому часто отдельные части кода распределяют по отдельным файлам, особенно когда эти части кода можно использовать в других программах на PHP.

Инструкция include

Инструкция **include** подключает в программу внешний файл с кодом php. Например, определим файл *factorial.php*:

```
<?php
function getFactorial($n)
{
    $result=1;
    for($i=1; $i <= $n; $i++)
        $result*=$i;
    return $result;
}
?>
```

Здесь происходит вычисление факториала. Теперь подключим данный файл в нашу программу:

```
<?php
include "factorial.php";

$a = 5;
$fact = getFactorial($a);
echo "Факториал числа $a равен $fact";
?>
```

В место определения инструкции include будет вставляться весь код из файла *factorial.php*. При этом вставка файла должна происходить до использования функции, определенной в этом файле.

Инструкция include_once

Использование инструкции include имеет недостатки. Так, мы можем в разных местах кода неумышленно подключить один и тот же файл, что при выполнении кода вызовет ошибки.

Чтобы исключить повторное подключение файла, вместо инструкции `include` надо применять инструкцию **include_once**

```
<?php
include_once "factorial.php";

$a = 5;
$fact = getFactorial($a);
echo "Факториал числа $a равен $fact";
?>
```

И теперь, если мы подключим этот же файл с помощью include_once еще где-нибудь ниже, то это подключение будет проигнорировано, так как файл уже подключен в программу.

Инструкции require и require_once

Действие инструкции **require** подобно инструкции include: она также подключает внешний файл, вставляя в программу его содержимое. Только теперь, если данный файл не будет найден, действие программы прекратится:

```
<?php
require "factorial.php";
?>
```

Так же если у нас в коде встретятся несколько инструкций `require`, которые подключают один и тот же файл, то интерпретатор выдаст ошибку. И так же чтобы избежать данной ситуации, надо использовать инструкцию require_once.

2.7. Массивы

Массивы предназначены для хранения наборов данных или элементов. Каждый элемент в массиве имеет свой уникальный ключ и значение. Итак, сохраним в массив список моделей телефонов:

```
<?php
$phones[0] = "Nokia N9";
$phones[1] = "Samsung Galaxy ACE II";
$phones[2] = "Sony Xperia Z3";
$phones[3] = "Samsung Galaxy III";

for($i=0;$i<count($phones);$i++)
   echo "$phones[$i] <br />";
?>
```

Здесь создается массив $phones из четыре элементов. Каждый элемент в массиве представляет собой пару **ключ - значение**. Так, первый элемент `$phones[0] = "Nokia N9"` имеет ключ - число 0, а значение - строку "Nokia N9". В таких массивах числовые ключи еще называются индексами.

С помощью функции **count()** можно узнать количество элементов в массиве. А благодаря тому, что ключи идут по порядку от 0 до 3, и зная размер массива, можно вывести элементы массивы в цикле `for`.

Данное создание массива будет также эквивалентно следующему:

```
<?php
$phones[] = "Nokia N9";
$phones[] = "Samsung Galaxy ACE II";
$phones[] = "Sony Xperia Z3";
$phones[] = "Samsung Galaxy III";
$num = count($phones);
for($i=0;$i<$num;$i++)
   echo "$phones[$i] <br />";
?>
```

Если не указывается ключ элемента, то PHP в качестве ключей использует числа. При этом нумерация ключей начинается с нуля, а каждый новый ключ увеличивается на единицу.

Зная ключ элемента в массиве, мы можем обратиться к этому элементу, получить или изменить его значение:

```
// получим элемент по ключу 1
$myPhone = $phones[1];
echo "$myPhone <br />";
// присвоение нового значения
$phones[1] = "Samsung X650";
echo "$phones[1] <br />";
```

Но в качестве ключей могут использоваться не только целые числа, но и строки:

```
<?php
$phones["nokia"] = "Nokia N9";
$phones["samsumg"] = "Samsung Galaxy III";
$phones["sony"] = "Sony Xperia Z3";
$phones["apple"] = "iPhone5";
echo $phones["samsumg"];
?>
```

Подобные массивы называют ассоциативными.

Оператор array

Выше был рассмотрен один способ создания массива. Но есть и другой, который предусматривает применение оператора `array()`.

```
<?php
$phones = array('iPhone', 'Samsung Galaxy S III', 'Nokia N9', 'Sony XPeria Z3');
echo $phones[1];
?>
```

Оператор `array()` принимает набор элементов. Здесь также явным образом не указаны ключи. Поэтому PHP автоматически нумерует элементы с нуля. Но мы также можем указать для каждого элемента ключ:

```
<?php
$phones = array("apple"=>"iPhone5", "samsumg"=>"Samsung Galaxy III",
"nokia" => "Nokia N9", "sony" => "Sony XPeria Z3");
echo $phones["samsumg"];
?>
```

Операция => позволяет сопоставить ключ с определенным значением.

Перебор ассоциативных массивов

Выше мы посмотрели, как с помощью цикла `for` вывести все элементы массива, где ключи заданы последовательно числами от 0 до 3. Однако с ассоциативными массивами это не работает. И для них в PHP предназначен специальный тип цикла – **foreach...as**:

```
<?php
$phones = array("apple"=>"iPhone5",
              "samsumg"=>"Samsung Galaxy III",
              "nokia" => "Nokia N9",
              "sony" => "Sony XPeria Z3");
foreach($phones as $item)
   echo "$item <br />";
?>
```

В цикле foreach из массива последовательно извлекаются все элементы и их значение помещается в переменную, указанную после ключевого слова **as**. В данном случае в переменную `$item` по очереди помещаются все четыре значения из массива $phones. Когда будет извлечен последний элемент из массива, цикл завершается.

Цикл `foreach` позволяет извлекать не только значения, но и ключи элементов:

```
<?php
$phones = array("apple"=>"iPhone5",
              "samsumg"=>"Samsung Galaxy III",
              "nokia" => "Nokia N9",
              "sony" => "Sony XPeria Z3");
foreach($phones as $key=>$value)
   echo "$key => $value <br />";
?>
```

Здесь при переборе элементов цикла в переменную `$key` будет передаваться ключ элемента, а в переменную `$value` - ее значение.

Альтернативу циклу `foreach` представляет использование функций **list** и **each**:

```
<?php
$phones = array("apple"=>"iPhone5",
              "samsumg"=>"Samsung Galaxy III",
              "nokia" => "Nokia N9",
              "sony" => "Sony XPeria Z3");
while (list($key, $value) = each($phones))
   echo "$key => $value <br />";
?>
```

Цикл while будет работать, пока функция `each` не вернет значение `false`. Функция `each` проходит по всем элементам массива $phones и получает его в виде массива, в который входят ключ и значение элемента. Затем этот массив передается функции list и происходит присваивание значения массива переменным внутри скобок. Когда функция `each` закончит перебор элементов массива $phones, она возвратит `false`, и действие цикла while будет завершено.

Многомерные массивы

В предыдущих примерах рассматривались только одномерные массивы, где значения элементов представляли числа, строки. Но в PHP массивы могут также быть многомерными, то есть такими, где элемент массива сам является массивом. Например, создадим многомерный массив:

```
<?php
$phones = array(
        "apple"=> array("iPhone5", "iPhone5s", "iPhone6") ,
        "samsumg"=>array("Samsung Galaxy III", "Samsung Galaxy
ACE II"),
        "nokia" => array("Nokia N9", "Nokia Lumia 930"),
        "sony" => array("Sony XPeria Z3", "Xperia Z3 Dual", "Xperia
T2 Ultra"));
foreach ($phones as $brand => $items)
{
   echo "<h3>$brand</h3>";
   echo "<ul>";
   foreach ($items as $key => $value)
   {
      echo "<li>$value</li>";
   }
   echo "</ul>";
}
?>
```

Чтобы обратиться к элементу данного массива, надо указать ключи в квадратных скобках. Например, обратимся к первому элементу в первом массиве. Так как ключ первого массива - "apple", а ключ первого элемента в первом массиве – число 0 (так как мы явным образом не указали ключи):

```
echo $phones["apple"][0];
```

Подобным образом можно получить второй элемент третьего массива:

```
echo $phones["nokia"][1];
```

Допустим, вложенные массивы также представляют ассоциативные массивы:

```
<?php
$technics = array(
        "phones" => array("apple" => "iPhone5",
                  "samsumg" => "Samsung Galaxy III",
                  "nokia" => "Nokia N9"),
      "tablets" => array("lenovo" => "Lenovo IdeaTab A3500",
                   "samsung" => "Samsung Galaxy Tab 4",
                    "apple" => "Apple iPad Air"));
foreach ($technics as $tovar => $items)
{
   echo "<h3>$tovar</h3>";
   echo "<ul>";
   foreach ($items as $key => $value)
   {
      echo "<li>$key : $value</li>";
   }
```

```
        echo "</ul>";
    }
    // присвоим одному из элементов другое значение
    $technics["phones"]["nokia"] = "Nokia Lumnia 930";
    // выведем это значение
    echo $technics["phones"]["nokia"];
    ?>
```

Операции с массивами

Функция `is_array()` проверяет, является ли переменная массивом, и если является, то возвращает `true`, иначе возвращает `false`. Например:

```
$isar = is_array($technics);
echo ($isar==true)?"это массив":"это не массив";
```

Функции `count()` и `sizeof()` получают количество элементов массива:

```
$number = count($technics);
// то же самое, что
// $number = sizeof($technics);
echo "В массиве technics $number элементов";
```

Функция `shuffle` перемешивает элементы массивы случайным образом:

```
$os = array("Windows 95", "Windows XP", "Windows Vista", "Windows 7", "Windows 8", "Windows 10");
shuffle($os);
print_r($os);
// один из возможных вариантов
// Array ( [0] => Windows 95 [1] => Windows 7 [2] => Windows Vista [3] => Windows XP [4] => Windows 10 [5] => Windows 8)
```

Функция `compact` позволяет создать из набора переменных ассоциативный массив, где ключами будут сами имена переменных:

```
<?php
$model = "Apple II";
$producer = "Apple";
$year = 1978;

$data = compact('model', 'producer', 'year');
print_r($data);
// получится следующий вывод:
// Array ( [model] => Apple II [producer] => Apple [year] => 1978 )
?>
```

Функция compact получает в скобках набор переменных. Каждая переменная указывается в кавычках без знака $. Результатом функции является новый массив.

Сортировка массивов

В PHP имеются два типа сортировки: сортировка строк по алфавиту и сортировка чисел по возрастанию/убыванию. Если сортируемые значения представляют строки, то они сортируются по алфавиту, если числа - то в порядке возрастания чисел. PHP по умолчанию самостоятельно выбирает тип сортировки.

Для сортировки по возрастанию используется функция **asort**:

```
<?php
$tablets = array("lenovo" => "Lenovo IdeaTab A3500",
                 "samsung" => "Samsung Galaxy Tab 4",
                 "apple" => "Apple iPad Air");
asort($tablets);

echo "<ul>";
foreach ($tablets as $key => $value)
{
    echo "<li>$key : $value</li>";
}
echo "</ul>";
?>
```

В данном случае значения массива представляют строки, поэтому PHP выберет сортировку по алфавиту. Однако с помощью дополнительного параметра мы можем явно указать интерпретатору PHP тип сортировки. Данный параметр может принимать три значения:

`SORT_REGULAR`: автоматический выбор сортировки

`SORT_NUMERIC`: числовая сортировка

`SORT_STRING`: сортировка по алфавиту

Укажем явно тип сортировки:

```
asort($tablets, SORT_STRING);
```

Чтобы отсортировать массив в обратном порядке, применяется функция **arsort**:

```
arsort($tablets);
```

Сортировка по ключам

Функция `asort` производит сортировку по значениям элементов, но также существует еще и сортировка по ключам. Она представлена функцией **ksort**:

```
ksort($tablets, SORT_STRING);
```

Сортировка по ключам в обратном порядке выполняется функцией **krsort()**:

```
krsort($tablets);
```

Естественная сортировка

Хотя описанные выше функции сортировки прекрасно выполняют свою работу, но их возможностей все-таки недостаточно. Например, отсортируем по возрастанию следующий массив:

```
<?php
$os = array("Windows 7", "Windows 8", "Windows 10");
asort($os);
print_r($os);
// результат
// Array ( [2] => Windows 10 [0] => Windows 7 [1] => Windows 8 )
?>
```

Так как значения представляют строки, то PHP сортирует по алфавиту. Однако подобная сортировка не учитывает числа и регистр. Поэтому значение "Windows 10" будет идти в самом начале, а не в конце, как должно было быть. И для решения этой проблемы в PHP есть функция **natsort()**, которая выполняет естественную сортировку:

```
<?php
$os = array("Windows 7", "Windows 8", "Windows 10");
natsort($os);
print_r($os);
// результат
// Array ( [0] => Windows 7 [1] => Windows 8 [2] => Windows 10)
?>
```

Если нам надо еще при этом, чтобы сортировка не учитывала регистр, то мы можем применить функцию **natcasesort()**:

```
natcasesort($os);
```

Передача параметров. Работа с формами. Обработка форм

Одним из основных способов передачи данных веб-сайту является обработка форм. Формы представляют специальные элементы разметки HTML, которые содержат в себе различные элементы ввода - текстовые поля, кнопки и т.д. И с помощью данных форм мы можем ввести некоторые данные и отправить их на сервер. А сервер уже обрабатывает эти данные.

Создание форм состоит из следующих аспектов:

- Создание элемента <form><form> в разметке HTML;
- Добавление в этот элемент одного или нескольких полей ввода;
- Установка метода передачи данных: GET или POST;
- Установка адреса, на который будут отправляться введенные данные.

Итак, создадим новую форму. Для этого определим новый файл *form.php*, в который поместим следующее содержимое:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
<h3>Вход на сайт</h3>
<form action="login.php" method="POST">
    Логин: <input type="text" name="login" /><br><br>
    Пароль: <input type="text" name="password" /><br><br>
    <input type="submit" value="Войти">
</form>
</body>
</html>
```

Атрибут `action="login.php"` элемента `form` указывает, что данные формы будет обрабатывать скрипт *login.php*, который будет находиться с файлом *form.php* в одной папке. А атрибут `method="POST"` указывает, что в качестве метода передачи данных будет применяться метод POST.

Теперь создадим файл *login.php*, который будет иметь следующее содержание:

```
<?php
$login = "Не известно";
$password = "Не известно";
if(isset($_POST['login'])) $login = $_POST['login'];
if (isset($_POST['password'])) $password = $_POST['password'];

echo "Ваш логин: $login  <br> Ваш пароль: $password";
?>
```

Чтобы получить данные формы, используется глобальная переменная **$_POST**. Она представляет ассоциативный массив данных, переданных с помощью метода POST. Используя ключи, мы можем получить отправленные значения. Ключами в этом массиве являются значения атрибутов `name` у полей ввода формы.

Так как атрибут `name` поля ввода логина имеет значение `login` (`<input type="text" name="login" />`), то в массиве $_POST значение этого поля будет представлять ключ "login": `$_POST['login']`

И поскольку возможны ситуации, когда поле ввода будет не установлено, например, при прямом переходе к скрипту: *http://localhost:8080/login.php*. В этом случае желательно перед обработкой данных проверять их наличие с помощью функции **isset()**. И если переменная установлена, то функция isset() возвратит значение `true`.

Необязательно отправлять данные формы другому скрипту, их можно обработать в том же файле формы. Для этого изменим файл *form.php* следующим образом:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
<div>
<?php
if(isset($_POST['login']) && isset($_POST['password'])){

    $login=$_POST['login'];
    $password = $_POST['password'];
    echo "Ваш логин: $login <br> Ваш пароль: $password";
}
?>
</div>
<h3>Вход на сайт</h3>
<form method="POST">
    Логин: <input type="text" name="login" /><br><br>
    Пароль: <input type="text" name="password" /><br><br>
    <input type="submit" value="Отправить">
</form>
</body>
</html>
```

Получение данных из строки запроса

Другим распространенным способом отправки данных на сервер является метод GET. Его сущность состоит в том, что данные передаются в адресной строке браузера.

Итак, создадим простой скрипт *get.php* со следующим содержимым:

```
<?php
$login = "не определен";
$age = "не определен";
if(isset($_GET['login'])){

    $login = $_GET['login'];
}
if(isset($_GET['age'])){

    $age = $_GET['age'];
}
    echo "Ваш логин: $login <br> Ваш возраст: $age";
?>
```

В PHP по умолчанию определен глобальный ассоциативный массив $_GET, который хранит все значения, передаваемые в строке запроса. Его действие подобно массиву $_POST: также по ключу мы можем получить передаваемое значение.

Теперь обратимся к этому скрипту, например, так: *http://localhost:8080 /get.php?login=mailcom&age=22*.

Все параметры передаются на сервер в следующей форме: *get.php?параметр1=значение1&параметр2=значение2&параметрN=значениеN*. Чтобы передать список параметров, после названия скрипта ставится знак вопроса, за которым идут наборы параметров и их значений. Название параметра и будет ключом в массиве $_GET. Наборы параметр-значения отделяются друг от друга знаком амперсанда (&).

Данные формы мы также можем передать через запрос GET. Для этого достаточно у формы указать атрибут method="get", и тогда все значения полей формы также будут передаваться через строку запроса:

```
<form method="GET">
    Логин: <input type="text" name="login" /><br><br>
    Пароль: <input type="text" name="password" />
<br><br>
    <input type="submit" value="Отправить">
</form>
```

Работа с полями ввода форм

Формы могут содержать различные элементы - текстовые поля, флажки, переключатели и т.д., обработка которых имеет свои особенности.

Обработка флажков

Флажки или чекбоксы `checkbox` могут находиться в двух состояниях: отмеченном (checked) и не отмеченном. Например:

```
Запомнить: <input type="checkbox" name="remember" checked="checked" />
```

Если флажок находится в неотмеченном состоянии, например:

```
Запомнить: <input type="checkbox" name="remember" />
```

то при отправке формы значение данного флажка не передается на сервер.

Если флажок отмечен, то при отправке на сервер для поля `remember` будет передано значение `on`:

```
$remember = $_GET['remember'];
```

Если нас не устраивает значение `on`, то с помощью атрибута `value` мы можем установить нужное нам значение:

```
Запомнить: <input type="checkbox" name="remember" value="1" />
```

Иногда необходимо создать набор чекбоксов, где можно выбрать несколько значений. Например:

```
ASP.NET: <input type="checkbox" name="technologies[]" value="ASP.NET" />
PHP: <input type="checkbox" name="technologies[]" value="PHP" />
RUBY: <input type="checkbox" name="technologies[]" value="Ruby" />
```

В этом случае значение атрибута name должно иметь квадратные скобки. И тогда после отправки сервер будет получать массив отмеченных значений:

```
$technologies = $_POST['technologies'];
foreach($technologies as $item) echo "$item<br />";
```

В данном случае переменная `$technologies` будет представлять массив, который можно перебрать и выполнять все другие операции с массивами.

Переключатели

Переключатели, или радиокнопки, позволяют сделать выбор между несколькими взаимоисключающими вариантами:

```
<input type="radio" name="course" value="ASP.NET" />ASP.NET <br>
<input type="radio" name="course" value="PHP" />PHP <br>
<input type="radio" name="course" value="Ruby" />RUBY <br>
```

На сервер передается значение атрибута `value` у выбранного переключателя. Получение переданного значения:

```
if(isset($_POST['course']))
{
    $course = $_POST['course'];
    echo $course;
}
```

Список

Список представляет элемент `select`, который предоставляет выбор одного или нескольких элементов:

```
<select name="course" size="1">
    <option value="ASP.NET">ASP.NET</option>
    <option value="PHP">PHP</option>
    <option value="Ruby">RUBY</option>
    <option value="Python">Python</option>
</select>
```

Элемент `<select>` содержит ряд вариантов выбора в виде элементов `<option>`.

Теперь получим выбранный элемент:

```
if(isset($_POST['course']))
{
    $course = $_POST['course'];
    echo $course;
}
```

Но элемент select также позволяет множественный выбор. И в этом случае обработка выбранных значений изменяется, так как сервер получает массив значений:

```
<select name="courses[]" size="4" multiple="multiple">
    <option value="ASP.NET">ASP.NET</option>
    <option value="PHP">PHP</option>
    <option value="Ruby">RUBY</option>
    <option value="Python">Python</option>
</select>
```

Такие списки имеют атрибут `multiple="multiple"`. Для передачи массива также указываются в атрибуте `name` квадратные скобки:`name="courses[]"`

Теперь получим в PHP выбранные значения:

```
<?php
if(isset($_POST['courses']))
{
    $courses = $_POST['courses'];
    foreach($courses as $item) echo "$item<br>";
}
?>
```

Пример обработки форм

Рассмотрим комплексный пример обработки форм, в котором объединим обработку различных элементов html. Определим следующую форму:

```
<h2>Анкета</h2>
<form action="input.php" method="POST">
<p>Введите имя:<br>
<input type="text" name="firstname" /></p>
<p>Форма обучения: <br>
<input type="radio" name="eduform" value="очно" />очно <br>
<input type="radio" name="eduform" value="заочно" />заочно
</p>
<p>Требуется общежитие:<br>
<input type="checkbox" name="hostel" />Да</p>
<p>Выберите курсы: <br>
<select name="courses[]" size="5" multiple="multiple">
    <option value="ASP.NET">ASP.NET</option>
    <option value="PHP">PHP</option>
    <option value="Ruby">RUBY</option>
    <option value="Python">Python</option>
    <option value="Java">Java</option>
</select></p>
<p>Краткий комментарий: <br>
<textarea name="comment" maxlength="200"></textarea></p>
<input type="submit" value="Выбрать">
</form>
```

Здесь простейшая стандартная форма ввода анкетных данных:

Теперь определим скрипт *input.php*, который будет обрабатывать эту форму:

```
<?php
if(isset($_POST['firstname']) && isset($_POST['eduform']) &&
   isset($_POST['comment']) && isset($_POST['courses']))
{
   $name = htmlentities($_POST['firstname']);
   $eduform = htmlentities($_POST['eduform']);
   $hostel = "нет";
   if(isset($_POST['hostel'])) $hostel = "да";
   $comment = htmlentities($_POST['comment']);
   $courses = $_POST['courses'];
   $output ="
   <html>
   <head>
   <title>Анкетные данные</title>
   </head>
   <body>
   Вас зовут: $name<br />
   Форма обучения: $eduform<br />
   Требуется общежитие: $hostel<br />
   Выбранные курсы:
   <ul>";
   foreach($courses as $item)
      $output.="<li>" . htmlentities($item) . "</li>";
   $output.="</ul></body></html>";
   echo $output;
}
else
{
   echo "Введенные данные некорректны";
}
?>
```

2.8. Упражнения на PHP

Упражнение 1. Проверка подключения веб-сервера

Перед работой необходимо убедиться, что php подключен и работает правильно. Для этого перейдите в папку *c:/localhost*, которую нужно создать для хранения документов, и добавьте в нее обычный текстовый файл. Переименуйте его в index.php и внесите в него следующее содержание:

```
<?php
phpinfo();
?>
```

В данном случае мы создали простейший скрипт, который выводит общую информацию о PHP. Теперь обратимся к этому скрипту, набрав в строке браузера адрес *http://localhost:8080/index.php.*

При обращении к сайту на локальном ПК в качестве адреса указывается *http://localhost*. Так как мы указали в качестве порта 8080, то также в адресе указывается через двоеточие порт. Если бы мы использовали 80-й порт, который используется по умолчанию, то его не надо было указывать.

Затем указывается имя ресурса, к которому идет обращение. В данном случае в качестве ресурса используется файл index.php. И так как в файле httpd.conf в качестве хранилища документов веб-сервера указан каталог *C:\localhost*, то именно в этом каталоге и веб-сервер будет производить поиск нужных файлов.

И поскольку выше при конфигурировании мы указали, что в качестве главной страницы может использоваться файл *index.php*, то мы можем также обратиться к этому ресурсу просто *http://localhost:8080/*

Упражнение 2. Работа с циклом

Откройте текстовый редактор Блокнот, наберите следующий код, а затем сохраните его под названием print.php в папке *c:/localhost*:

```
<?php
for ($i = 1; $i < 10; $i++)
{
   if($i==5)
   {
      continue;
   }
echo "Квадрат числа $i равен " , $i * $i , "<br/>";
}
?>
```

Для добавления выражений PHP на страницу используются теги <?php ..... ?>, между которыми идут инструкции на языке PHP. Каждое отдельное выражение PHP должно завершаться точкой с запятой. В данном случае выполняется текст с параметром, внутри которого есть условный оператор и оператор вывода на экран квадратов чисел.

Теперь обратимся к этому скрипту, набрав в строке браузера адрес *http://localhost:8080/print.php*. Обратите внимание на вывод чисел. Попробуйте изменить диапазон вычисления с 10 до 100.

Упражнение 3. Работа с таблицей

Откройте текстовый редактор Блокнот, наберите следующий код, а затем сохраните его под названием table.php в папке *c:/localhost*:

```
<html>
<body>
<table border="1">
 <?php
 $cols = 10;
 $rows = 10;
        for($tr=1; $tr<=$rows; $tr++){
        echo "<tr>";
                for($td=1; $td<=$cols; $td++){
                echo "<td>", $tr * $td, "</td>";
                }
        echo "</tr>";
        }
 ?>
 </table>
</body>
</html>
```

Теперь обратимся к этому скрипту, набрав в строке браузера адрес *http://localhost:8080/table.php*. Обратите внимание на числа в ячейках таблицы. Попробуйте изменить код так, чтобы выводилась не таблица умножения, а таблица сложения.

Упражнение 4. Работа с числовым массивом

Откройте текстовый редактор Блокнот, наберите следующий код, а затем сохраните его под названием mas1.php в папке *c:/localhost*:

```
<?php
 $i=0; //Индекс 1 элемента
 while ($i < count($num))
 {
  echo $num[$i].' ';
  $i++;
  //Проверка если $i равен числу элементов в массиве
  //тогда выводим последний элемент и возвращаем указатель
  if ($num[$i] == count($num))
  {
   echo $num[$i];
   reset ($num);
   echo '<br />'."Конец массива";
   exit();
  }
 }
?>
<?php
 // Определяем массив
 $arr = array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
 // Генерируем случайный индекс массива
 $index = rand(0,count($arr) - 1);
 // Выводим случайный элемент массива
 echo $arr[$index];
?>
```

Теперь обратимся к этому скрипту, набрав в строке браузера адрес *http://localhost:8080/mas1.php*.

Упражнение 5. Работа с ассоциативным массивом

Откройте текстовый редактор Блокнот, наберите следующий код, а затем сохраните его под названием mas2.php в папке *c:/localhost*:

```
<?php
$capitals = array("Russia"=>"Moscow", "France"=>"Paris", "Italy"=>"Rome");
for (reset($capitals); $k = key($capitals); next($capitals)) {
   echo "Столица $k - $capitals[$k] </br>";
}
?>
```

Теперь обратимся к этому скрипту, набрав в строке браузера адрес *http://localhost:8080/mas2.php*.

Откройте текстовый редактор Блокнот, наберите следующий код, а затем сохраните его под названием mas3.php в папке *c:/localhost*:

```
<?php
 $arr = array( array(‘Вася’, ‘слесарь’, 2500 ),
array(‘Миша’,’строитель’, 3000),
array(‘Андрей’, ‘шофер’, 2700));

for ($i = 0; $i < 3; $i++)
 {
  for ($j=0; $j <3; $j++)
  {
    echo ' | '.$arr[$i][$j];
  }
  echo '<br />';
 }
?>
```

Теперь обратимся к этому скрипту, набрав в строке браузера адрес *http://localhost:8080/mas3.php*.

Откройте текстовый редактор Блокнот, наберите следующий код, а затем сохраните его под названием mas4.php в папке *c:/localhost*:

```
<?php
$цена = array ("помидоры" => 15, "огурцы" => 12);
foreach ($цена as $овощи => $руб)
 {
 echo  "$овощи стоят $руб руб.<br>";
 }
?>
```

Теперь обратимся к этому скрипту, набрав в строке браузера адрес *http://localhost:8080/mas4.php*.

Откройте текстовый редактор Блокнот, наберите следующий код, а затем сохраните его под названием mas5.php в папке *c:/localhost*:

```
<?php
$данные = array (
        "Иванов" => array ("рост" => 174, "вес"  => 68),
        "Петров" => array ("рост" => 181, "вес"  => 90),
        "Сидоров" => array ("рост" => 166, "вес"  => 73));
foreach ($данные as $фамилия => $данные1)
 {
 echo  "<br>$фамилия:<br>";
 foreach ($данные1 as $параметр => $pp)
  {
  echo  "$параметр = $pp<br>";
  }
 }
?>
```

Теперь обратимся к этому скрипту, набрав в строке браузера адрес *http://localhost:8080/mas5.php*.

Упражнение 6. Первый сайт на PHP

Перейдите к ранее созданному каталогу *c:/localhost*:, который будет хранить все документы сайта. Создайте текстовый файл, который назовем *index.html*. Откроем его в текстовом редакторе и добавим в него следующий код:

```
<!DOCTYPE html>
<html>
<head>
<title>Первый сайт на PHP</title>
<meta charset="utf-8">
</head>
<body>
<h2>Введи свои данные:</h2>
<form action="display.php" method="POST">
<p>Введите имя: <input type="text" name="firstname" /></p>
<p>Введите фамилию: <input type="text" name="lastname" />
</p>
<input type="submit" value="Отправить">
</form>
</body>
</html>
```

Код html содержит форму с двумя текстовыми полями. При нажатии на кнопку данные этой формы отсылаются скрипту display.php, так как он указан в атрибуте action.

В коде php мы получаем данные формы и выводим их на страницу.

Теперь создадим этот скрипт, который будет обрабатывать данные. Добавим в папку *C:\localhost* новый текстовый файл. Переименуем его в display.php. По умолчанию файлы программ на php имеют расширение *.php*. Итак, добавим в файл display.php следующий код:

```
<!DOCTYPE html>
<html>
<head>
<title>Первый сайт на PHP</title>
<meta charset="utf-8">
</head>
<body>
<?php
$name = $_POST["firstname"];
$surname = $_POST["lastname"];
echo "Ваше имя: <b>".$name . " " . $surname . "</b>";
?>
</body>
</html>
```

Здесь уже в разметке html идут вкрапления кода PHP.

В данном случае у нас три выражения. Два из них получают переданные данные формы, например, $name = $_POST["firstname"];.

$name – это переменная, которая будет хранить некоторое значение. Все переменные в PHP предваряются знаком $. И так как форма на странице index.html использует для отправки метод POST, то с помощью выражения $_POST["firstname"] мы можем получить значение, которое было введено в текстовое поле с атрибутом name="firstname". И это значение попадает в переменную $name.

С помощью оператора echo можно вывести на страницу любое значение или текст, которые идут после оператора. В данном случае (echo "Ваше имя: ".$name . " " . $surname . "") с помощью знака точки текст в кавычках соединяется со значениями переменных $name и $surname и выводится на страницу.

Теперь обратимся к форме ввода, перейдя по адресу *http://localhost:8080*:

Введем какие-нибудь данные и нажмем на кнопку «Отправить».

Итак, у нас сработал скрипт *display.php*, который получил и вывел отправленные данные на страницу.

2.9. Задания для самостоятельного выполнения

Задание № 1. Циклы

1. Вывести на экран все нечетные n-значные числа ($1 < n \leq 4$).

2. Вывести таблицу умножения на p, используя различные циклы.

3. Вывести на экран таблицу умножения.

4. Вывести на экран таблицу сложения чисел от 1 до 9.

5. Написать программу, выводящую на экран таблицу, в которой количество строк и столбцов задается пользователем. В каждой ячейке выводится ее порядковый номер.

6. Вывести на экран таблицу, ячейки которой закрашены в разные цвета, а номера цветов отображаются в соответствующих ячейках. Номера цветов изменяются от 555555 до 999999 с шагом 1111

7. Вывести на экран строку разных размеров и цветов. Цвет изменяется от 111111 до 999999 с шагом 111111.

8. Вывести на экран таблицу сложения чисел от 10 до 20.

9. Вывести на экран один и тот же текст шестнадцатью случайными цветами.

10. Получить сумму n случайных чисел из диапазона [-50; 50]. Вывести суммируемые числа (отрицательные — одним цветом; положительные — другим; нули — третьим) и сумму.

11. Вывести на экран таблицу размером $m \times n$, ячейки которой закрашены в случайные цвета.

12. Вывести заданное натуральное число в системах счисления от 2 до n включительно ($n <= 36$).

13. Вывести на экран маркированный список из n элементов, используя различные маркеры.

14. Вывести на экран таблицу размером $m \times n$, ячейки которой содержат рисунки (для удобства рисунки можно хранить в файлах с именами 1.jpg, 2.jpg и т.д.).

15. Вывести на экран квадратную таблицу заданного размера, диагональные ячейки которой окрашены в случайные цвета.

Задание № 2. Массивы

1. Создайте многомерный массив, содержащий названия фильмов, организованных по жанрам: ассоциативный массив, в котором имена полей будут разными жанрами («комедия», «мелодрама», «детектив» и др.), а элементами — названия фильмов. Выведите информацию.

2. Создайте ассоциативный массив, аналогичный телефонному справочнику. Отсортируйте массив по фамилиям абонентов в алфавитном порядке.

3. Создайте ассоциативный многомерный массив, содержащий **информацию о пользователях** (ФИО, возраст, количество посещений страницы). Выведите всю информацию, начиная с пользователей, у которых количество посещений страницы больше.

4. Создайте массив, содержащий сведения **об учениках класса** (фамилия, рост, вес, средний балл). Найдите самого высокого ученика и выведите всю информацию о нем.

5. Создайте массив, содержащий сведения **о ваших друзьях**. Отсортируйте его по фамилиям друзей в алфавитном порядке и выведите всю информацию.

6. Создайте массив, содержащий сведения **о ваших друзьях**. Отсортируйте его по возрасту друзей и выведите всю информацию.

7. Создайте массив, содержащий сведения **о продукции фирмы**: номер товара, название, цена. Отсортируйте массив по названиям в алфавитном порядке. Среди товаров с одинаковым названием сначала идут более дешевые.

8. Создайте массив, содержащий сведения **о картинках**: местонахождение и имя файла, хранящего картинку, ее размеры, название. Выведите все картинки на экран с полной информацией о них.

9. Создайте многомерный массив, содержащий названия музыкальных произведений, организованных по жанрам: ассоциативный массив, в котором имена полей будут разными жанрами («рок», «поп», «джаз» и др.), а элементами — названия песен. Выведите информацию.

10. Описать массив-расписание, содержащий

- день недели;
- количество пар в этот день;
- время начала и конца пары;
- название предмета;

- фамилия преподавателя.

Вывести полную информацию о занятиях, относящихся к предметной области «Информатика».

11. В библиотеке имеются книги, газеты, журналы. Для каждого печатного издания указать

- название;
- год выпуска (для книги), дату выпуска (для газет и журналов);
- автора (для книги), редактора (для газеты), редколлегию (для журнала);
- объем.

Вывести информацию об изданиях, вышедших в заданном году.

12. Опишите массив, содержащий информацию **о движении электропоездов** из вашего города: направление; время отправления электропоездов, время в пути до конечного пункта, стоимость билетов по зонам. Вывести перечень электропоездов, следующих в заданном направлении.

13. Описать массив **экзаменационная ведомость** (предмет, номер группы, номер зачетной книжки, фамилия, имя, отчество студента, его оценки по итогам текущей сессии). Определить отличников, хорошистов, троечников и двоечников.

14. Описать массив **служащий**, включающий имена, фамилии, отчества служащих, даты рождения, полученное образование, домашние адреса, профессии. Определить имена людей с высшим образованием. Выдать данные о служащем, который имеет ту или иную профессию.

15. При поступлении в университет лица, получившие оценку «неудовлетворительно» на первом экзамене, ко второму экзамену не допускаются. Считая фамилии абитуриентов и их оценки после первого экзамена исходными данными, составить список абитуриентов, допущенных ко второму экзамену.

16. Создайте многомерный массив, содержащий названия книг, организованных по жанрам: ассоциативный массив, в котором имена полей будут разными жанрами («детектив», "женский роман", "классика" и др.), а элементами — названия книг. Выведите информацию.

17. Описать массив **служащий**, включающий имя, фамилию, отчество служащего, дату рождения, образование, домашний адрес, профессию. Определить имена людей с высшим образованием. Выдать данные о служащем, который имеет ту или иную профессию.

18. **Массив содержит сведения об** учителях школы. Распечатать список тех учителей, которые преподают математику и информатику, указать стаж их работы и недельную нагрузку.

19. Массив содержит сведения **о работниках предприятия**. Найти тех, чья заработная плата за месяц является ниже средней по предприятию, а также распечатать список тех, кто проработал на предприятии более 10 лет с указанием их фамилии, зарплаты, стажа работы и должности.

20. Массив содержит сведения **о спортсменах**. Распечатать данные о тех из них, кто занимается плаванием. Указать возраст, их стаж занятий спортом.

Задание № 3. Функции

1. Написать функцию, возвращающую массив из n случайных чисел.

2. Вывести все счастливые билеты из отрезка $[M, N]$, где M и N — шестизначные числа.

3. Написать функцию, которая находит номера максимального и минимального элементов массива, а также среднее арифметическое всех элементов массива.

4. Вывести на экран решето Эратосфена. Это таблица, в которой содержатся натуральные числа от a до b, где составные числа перечеркнуты. Использовать функцию, определяющую, является ли число простым.

5. Написать функцию, создающую таблицу, количество строк и столбцов которой принимается в качестве аргументов.

6. Написать функцию, возвращающую сумму всех элементов целочисленного массива.

7. Написать функцию, выводящую строку заданного размера шрифта. Строка и размер шрифта задаются в качестве аргументов функции.

8. Написать функцию, выводящую строку заданного цвета шрифта. Строка и цвет шрифта задаются в качестве аргументов функции.

9. Написать функцию для форматированного вывода текущей даты и дня недели.

10. Написать функцию, возвращающую произведение всех элементов целочисленного массива.

11. Описать функцию, которая удаляет из заданной строковой величины все лишние пробелы. Пробелы считаются лишними, если их подряд идёт более

двух, если они стоят в конце строки после последней точки, если стоят после открывающегося парного знака препинания.

12. Даны натуральные числа n и k, $n > 1$. Напечатать k десятичных знаков числа $1/n$. Программа должна использовать только целые переменные.

13. Найти все натуральные числа, не превосходящие заданного n, которые делятся на каждую из своих цифр.

14. Из заданного числа вычли сумму его цифр. Из результата вновь вычли сумму его цифр и т.д. Через сколько таких действий получится нуль?

15. Написать функцию, формирующую таблицу символов и их кодов из заданного диапазона.

Задание № 4. Работа с формами

1. Написать программу-вычислитель, которая позволяет пользователю передать два числа и указать операцию, выполняемую над ними.

2. Составить программу, благодаря которой пользователь может выбрать цвет и размер шрифта из предложенного списка.

3. Написать программу, обрабатывающую ввод анкетных данных клиента и реагирующую соответствующим образом. Например, программа может выводить строку «Здравствуйте, Иванов Петр Федорович! Спасибо за заказ! Мы рады приветствовать в вашем лице клиента нашей фирмы» или «Здравствуйте, Симонова Инна Леонидовна! Спасибо за заказ! Так как сумма вашего заказа превысила 5000 рублей, Вы получаете 10 %-ную скидку». Анкетные данные: ФИО, возраст, сумма заказа.

4. Составить программу, благодаря которой пользователь может выбрать цвет фона из предложенного списка.

5. Создать форму для приема сообщения посетителя и выводить его сообщение на другой странице.

6. Создать форму со списком товаров на продажу и выводить информацию о заказе посетителя на другой странице.

7. Составить программу, благодаря которой пользователь может выбрать язык (русский или английский, например), на котором отображается содержимое страницы.

8. Создать форму со списком предоставляемых услуг вашей туристической фирмой. После выбора пользователем какой-нибудь услуги выводится более полная информация о ней.

9. Составить программу, благодаря которой пользователь может выбрать цвет шрифта из предложенного списка.

10. Составить программу, благодаря которой пользователь может выбрать фоновую картинку из предложенного списка.

11. Составить программу, благодаря которой пользователь может выбрать стиль оформления списков из предложенного набора.

12. Составить программу, благодаря которой пользователь может выбрать размер шрифта текста из предложенного списка.

13. Написать программу отсчета дней до дня рождения. Посетитель вводит день, месяц и год рождения, в результате выводится сообщение о том, сколько дней осталось до дня его рождения.

14. Написать программу подсчета количества дней между двумя датами, указанными пользователем.

15. Написать программу для заполнения анкеты-резюме при приеме на работу.

16. Составить программу, благодаря которой пользователь может выбрать шрифт из предложенного списка.

17. Составить программу, благодаря которой пользователь может выбрать оформление таблицы из предложенного списка.

18. Создать форму со списком товаров на продажу и выводить информацию о заказе посетителя в текстовый файл.

19. Написать программу отсчета дней до праздника. Посетитель вводит дату праздника, в результате выводится сообщение о том, сколько дней осталось до этой даты.

20. Составить программу, благодаря которой пользователь может выбрать оформление ссылок из предложенного набора.

Задание № 5. Строки

1. Создать массив из целых и вещественных чисел. Вывести все его элементы в поля шириной 20 символов, преобразовывая их в вещественные значения с точностью в 2 знака после запятой.

2. Создать массив из целых и вещественных чисел. Вывести все его элементы в поля шириной 15 символов, преобразовывая их в целые значения.

3. Написать сценарий, который контролирует ввод электронного адреса. Если в электронном адресе не встречается символ "@", то выводится соответствующее сообщение и предлагается повторный ввод.

4. Отобразить на экране содержимое ассоциативного массива (содержит информацию о пользователях (ФИО, возраст, количество посещений страницы). Выведите всю информацию на браузер, начиная с пользователей, у которых количество посещений страницы больше), используя форматированный вывод.

5. Вывести на экран таблицу Пифагора (таблицу умножения), используя форматированный вывод.

6. Определить количество слов во введенном сообщении пользователя и вывести соответствующее сообщение.

7. Тэг курсива. Дан текст, в котором встречаются структуры "" и "". Заменить каждое вхождение "" на "<курсив>", а каждое вхождение "" на "<конец курсива>". **Замечание**. В программе следует учесть, что буква «i» может быть как строчной, так и прописной.

8. Дан текст. Напишите программу, определяющую процентное отношение строчных и прописных букв к общему числу символов в нем.

9. Дан текст. Определите, каких букв (строчных или прописных) в нем больше, и преобразуйте следующим образом: если больше прописных букв, чем строчных, то все буквы преобразуются в прописные; если больше строчных, то все буквы преобразуются в строчные; если поровну и тех, и других — текст остается без изменения.

10. Дана строка. Если она представляет собой запись целого числа, то вывести 1; если вещественного (с дробной частью), то вывести 2; если строку нельзя преобразовать в число, то вывести 0.

11. Дана строка, состоящая из русских слов, разделенных пробелами (одним или несколькими). Вывести строку, содержащую эти же слова (разделенные одним пробелом), но расположенные в обратном порядке.

12. Дана строка-предложение. Зашифровать ее, поместив вначале все символы, расположенные на четных местах, а затем, в обратном порядке, все

символы, расположенные на нечетных местах (например, строка "Программа" превратится в "ргамамроП")

13. Дано число в двоичной системе счисления. Проверить правильность ввода этого числа (в его записи должны быть только символы 0 и 1). Если число введено неверно, повторить ввод. При правильном вводе перевести число в десятичную систему счисления.

14. Двумерный массив $n \times m$ содержит некоторые буквы русского алфавита, расположенные в произвольном порядке. Написать программу, проверяющую, можно ли из этих букв составить данное слово S. Каждая буква массива используется не более одного раза.

15. В строке содержится запись арифметического выражения. Какие арифметические операции использованы в выражении?

16. В строке содержится запись арифметического выражения. Какие цифры есть в выражении?

17. В строке содержится запись арифметического выражения. Каких цифр нет в выражении?

18. В строке содержится запись арифметического выражения. Есть ли в записи выражения скобки?

19. Слова в тексте разделены пробелами. Какие символы есть в каждом слове?

20. Слова в тексте разделены пробелами. Какие символы встречаются в одном и только в одном слове?

Библиографический список

1. Афонин, С. М. PHP для начинающих. Обучение и решение задач: [функции для работы с базами данных, обзор программного обеспечения, графическая библиотека GD2, разработка интернет-приложений, полный спектр возможностей языка]. М.: NT Press, 2007. 253 с.

2. Дригалкин В.В. HTML в примерах. Как создать свой Web-сайт. М.: Издательский дом «Вильямс», 2004. 192 с.

3. Рева О.Н. JavaScript. Просто как дважды два. М.: Эксмо, 2006. 256 с.

4. Сайт о программировании. [Электронный ресурс]. URL: https://metanit.com/web/php/3.4.php (дата обращения: 01.09.2018).

5. Шестаков А.П. Учителям информатики и математики и их любознательным ученикам (дидактические материалы по информатике и математике). [Электронный ресурс]. URL: http://comp-science.narod.ru/ (дата обращения: 01.09.2018).

Учебное издание

Василюк Надежда Николаевна

ЯЗЫКИ ПРОГРАММИРОВАНИЯ
Основы web-программирования

Учебное пособие

Редактор *Л. Л. Савенкова*
Корректор *Л. И. Иванова*
Техническая подготовка материалов: *Н. Н. Василюк*

Объем данных 2,57 Мб
Подписано к использованию 29.05.2019

Размещено в открытом доступе
на сайте www.psu.ru
в разделе НАУКА / Электронные публикации
и в электронной мультимедийной библиотеке ELiS

Издательский центр
Пермского государственного
национального исследовательского университета
614990, г. Пермь, ул. Букирева, 15