МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Автоматизированные системы управления»

ПРОГРАММИРОВАНИЕ СЕТЕВЫХ ПРИЛОЖЕНИЙ

Методические рекомендации к лабораторным работам для студентов специальности 1-28 01 02 «Электронный маркетинг» очной и заочной форм обучения

Могилев 2020

УДК 004.43
ББК 32.973.26-018.1
П78

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Автоматизированные системы управления» «18» мая 2019 г., протокол № 10

Составитель канд. техн. наук, доц. Э. И. Ясюкович

Рецензент Ю. С. Романович

Методические рекомендации содержат основные базовые теоретические сведения, некоторые приемы реализации задач, а также практические задания для выполнения лабораторных работ по всем темам курса.

Учебно-методическое издание

ПРОГРАММИРОВАНИЕ СЕТЕВЫХ ПРИЛОЖЕНИЙ

Ответственный за выпуск	А. И. Якимов
Корректор	Т. А. Рыжикова
Компьютерная верстка	Н. П. Полевничая

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 26 экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, Могилев.

Содержание

Введение

Целью изучения дисциплины «Программирование сетевых приложений» является приобретение специальных знаний, умений и навыков, необходимых менеджеру специальности «Электронный маркетинг» для работы с интернет-приложениями.

Язык *html* (Hyper Text Markup Language) используется для добавления разметки в обычный текст, позволяет создавать статические и динамические сайты и является языком, описывающим структуру и семантику веб-документов.

В современных технологиях веб-программирования широко используются каскадные таблицы стилей (Cascading Style Sheets – CSS), которые влияют на отображение страниц в окнах браузеров (цвета, шрифты, фоновые изображения, интервалы между строками, отступы, границы, анимация элементов). Благодаря CSS можно производить изменения, относящиеся ко всем страницам сайта, редактируя при этом лишь один единственный файл – таблицы стилей.

В веб-программировании используются языки программирования, такие как *JavaScript*, PHP, *Python* и другие.

Для упрощения процесса разработки *JavaScript* скриптов, разработана и постоянно пополняется добровольцами библиотека *jQuery*. Основная цель создания этой библиотеки состоит в предоставлении возможности создания многоразовых фрагментов кода *JavaScript*, позволяющих упростить процесс создания *html*-документах.

Методические рекомендации предназначены для изучения основ программирования сетевых приложений и содержат 17 лабораторных работ. В каждой работе излагается краткий теоретический материал по вопросам соответствующей темы, предлагаются технологии решения некоторых типовых задач, приводятся задания для выполнения работ, а также контрольные вопросы.

Первые лабораторные работы методических рекомендаций посвящены изучению языка *html* и каскадных таблиц стилей CSS. Следующие ориентированы на изучение основ языка *JavaScript* и библиотеки *jQuery*, языка PHP, а две последние – технологий программирования в среде многозадачной операционной системы *Андроид* для смартфона, построенной на основе высокопроизводительной ОС Linux.

Каждая лабораторная работа рассчитана на 2 часа, а ее цель соответствует названию. Выполнение работ производится в следующем порядке:

1) ознакомиться с теоретическими положениями работы;

2) выбрать из таблицы «Варианты заданий» согласно варианту, указанному преподавателем, задания и исходные данные к работе, выполнить задания и оформить отчет, содержащий постановку задачи, ход выполнения и выводы.

1 Лабораторная работа № 1. Разработка веб-страниц на языке html

Для создания веб-страниц используется язык разметки гипертекста *html* (Hyper Text Markup Language), содержащий специальные теги, представляющие собой последовательности, заключенные между символами «<» и «>».

Структура веб-страницы, называемой также *html*-документом, имеет вид:

```
<!DOCTYPE html>
<html>
    <head>
        <title>простое название</title>
    </head>
    <body>
        содержание страницы
    </body>
</html>
```

В приведенной веб-странице строка <!DOCTYPE html> – это не *html*-тег, а инструкция браузеру о версии языка разметки *html*-документа.

Для формирования различных элементов веб-страниц на языке *html* используются теги, например, для создания заголовков – теги *h1…h6*. Списки на языке *html* создаются тегами *<ol>* (нумерованные) или *<ul>* (ненумерованные), а их элементы – тегами *<li>*.

Для обозначения элементов ненумерованных списков используются маркеры. Атрибут *type* тега *<ul>* позволяет изменить вид маркера и имеет следующие значения: *circle* – ○; *disk* – ●; *square* – ■; *A* – A, B, C; *a* – a, b, c; *I* – I, II, III; *i* – i, ii, iii; *1* – 1, 2, 3.

Таблицы в *html* создаются тегом *<table>* и состоят из строк и ячеек, которые задаются с помощью тегов *<tr>* и *<td>*. Атрибут *colspan* тега *<tr>* позволяет объединить ячейки столбцов, а с помощью атрибута *rowspan* тега *<td>* – ячейки строк.

Для отображения границ таблицы используется атрибут *border*, а с помощью тега *<th>* можно создать табличный заголовок.

Основой гипертекстовых документов являются ссылки, позволяющие организовать переход с одной веб-страницы на другую с помощью тега *<a>*:

```
<a href="url">текст ссылки</a>
```

Здесь атрибут *href* определяет *url* адрес документа, на который следует перейти (например, "*cat.html*"), а содержимое контейнера *<a>* является ссылкой.

Вставка изображений. Для добавления в *html*-документ изображений используется тег *<img>*, атрибут *src* которого должен содержать адрес картинки, которая должна быть отображена. Например:

*,

где *align* – выравнивание по правому, левому, верхнему, нижнему краям изображения;

border = 3 – обрамление в 3px;

hspace = 30 – вставка пустой области в 30px вокруг изображения;

width =110, *height* = 200 – изменение размеров изображения.

Для вставки звукового объекта используется тег

*<embed src = "*имя_файла.*wav width = " " height = " ">,*

где *width* и *height* – атрибуты для задания размеров экранных элементов.

Тег * *. Является блочным элементом и предназначен для выделения фрагмента документа с целью изменения вида содержимого. Как правило, вид блока управляется с помощью стилей.

Тег **. Предназначен для определения строчных элементов документа. В отличие от блочных элементов, таких как *<table>*, *<p>* или * *, с помощью тега ** можно выделить часть информации внутри других тегов и установить для нее свой стиль. Например, внутри абзаца, который создается с помощью тега *<p>*, можно изменить цвет и размер первой буквы, если добавить начальный и конечный тег ** и определить для него стиль текста.

Порядок выполнения работы.

1 Создать *html*-документ, содержащий две страницы. Вид первой страницы представлен на рисунке 1.1, ее *html*-код – на листинге 1.1, имя ее файла – «*Ваше Имя».html*, вторая страница – *«*Элемент гиперссылки, указанный в колонке 7 таблицы 1.1*».html*.

Рисунок 1.1 – Вид *html*-страницы

Листинг 1.1 – *html*-код веб-страницы

<!DOCTYPE html>
<html>

```
<head>
   <title>Табличная верстка</title>
</head>

<body>
<table border="1" cellpadding="0" cellspacing="0" width="100%">
<tr><th colspan=2>шапка сайта (логотип, слоган, телефон)</th></tr>
<tr><th width="20%">навигация</th><th width="80%">заголовок</th></tr>
<tr><td width="20%"><ul>
<li><a href="index.html" title="Ссылка 1">Ссылка 1</a></li>
<li><a href="index.html" title="Ссылка 2">Ссылка 2</a></li>
<li><a href="index.html" title="Ссылка 3">Ссылка 3</a></li>
</ul></td>
<td width="80%">контент</td>
</tr>
<tr><td colspan=2>Низ сайта (баннеры, счетчики, информация)</td></tr>
</table>
</body>
</html>
```

2 С помощью тега *span* поменять цвет двух первых строк заголовка *hi* левого элемента «Меню» веб-страницы на цвет, указанный в колонке 3.

3 При щелчке мышью по тексту «Низ сайта» цвет главной рамки сайта должен измениться на указанный в колонке 3, а по тексту в «Шапка» должно запускаться обтекание рамки вокруг элемента «Шапка» в направлении, указанном в колонке 8.

4 На второй странице построить таблицу с помощью тега *div*, состоящую из двух ячеек. В первой ячейке – записать Ваши фамилию, имя, отчество, а во второй – номер Вашего телефона.

Таблица 1.1 – Варианты заданий

Номер вари-анта	Выравни-вание «Текста страницы» по краю	Толщина рамки вокруг «Шапка», «Текст страницы», px, цвет	Ширина левого, правого меню, %		Теги *hi* заголовков меню	Элемент гиперссылки «Низ сайта»
1	2	3	4	5	6	7
1	Верхнему	4, 2, Gray:#808080	15	20	h2	Новости
2	По центру	3, 2, Red: #FF0000	16	19	h3	Погода
3	Нижнему	4, 3, Coral #FF7F50	17	18	h4	Товары
4	Правому	3, 2 Yellow:#FFFF00	18	17	h5	Услуги
5	Левому	5, 3, Indigo:#4B0082	19	16	h6	Работа
6	Верхнему	5, 2,Brown#A52A2A	20	15	h2	Отдых
7	По центру	5, 4, Lime:#00FF00	15	20	h3	Новости
8	Нижнему	3, 3, Blue:#0000FF	16	19	h4	Погода
9	Правому	4, 2, Peru:#CD853F	17	18	h5	Товары
10	Левому	4, 3, Navy:#000080	18	17	h6	Услуги
11	Верхнему	3, 2, Aqua:#00FFFF	19	16	h2	Работа
12	По центру	4, 5, Purple:#800080	20	15	h3	Отдых

Контрольные вопросы

1 Какие редакторы используются для редактирования веб-страниц?
2 Какое расширение имеет файл *html*?
3 Как вставить рисунок в *html*-документ?
4 Из каких разделов состоит *html*-документ?
5 Как изменить размер и цвет текста в *html*-документе?
6 Как вставить рисунок в ячейку таблицы *html*-документа?
7 Какие списки используются в *html*-документах?

2 Лабораторная работа № 2. Использование CSS при разработке веб-страниц

Для создания и изменения стиля элементов веб-страниц и пользовательских интерфейсов используются каскадные таблицы стилей *SCC* (*Cascading Style Sheets*).

Таблица стилей – это набор правил, которые применяются к *html*-тегам. При этом каждое правило состоит из селектора и блока определения.

Селектор – это тег *html*, сообщающий браузеру, какой элемент необходимо форматировать, а определение – его форматирующие команды, указывающие свойства и значения, которые указываются в фигурных скобках после селектора и разделяются двоеточием. В конце каждого определения селектора ставится точка с запятой, например:

```
p {color: #FF0000;}
```

где *p* – это селектор;
color: #FF0000 – определение.

Встраивать таблицы стилей в документ можно двумя способами. Первый способ: используется тег параметр *style*, который не требует селектора:

```
<p style="background-color:#003333;">
```

Второй способ: свойство указывается в заголовке документа между тегами *<style>….</style>*.

По методам добавления стилей в документ различают три их вида: внутренние, глобальные и внешние (связанные).

Внутренние стили определяются атрибутом *style* конкретных тегов и действуют только на определенные их элементы. Например:

```
<p style="color:blue">Абзац с текстом синего цвета</p>
```

Внутренние стили не рекомендуется использовать слишком часто, т. к. веб-документ оказывается перегруженным кодом, что затрудняет изменение его внешнего вида.

Глобальные стили являются универсальным средством, позволяющим не только оперативно изменять внешний вид веб-страницы, но и бороться с перегруженностью документа оформительскими тегами. Но прописывать глобальные стили необходимо на каждой странице сайта.

Глобальные стили CSS располагаются в контейнере *<style>...</style>* раздела *<head>...</head>*:

```
<head>
   .....
   <style type="text/css">
      p {color:#808080;}
   </style>
   .....
</head>
```

Атрибут *type="text/css"*, ранее обязательный для тега **<*style*>**, в стандарте *html5* можно опускать.

Внешние (связанные) стили определяются в отдельном файле с расширением *.css*. Внешние стили позволяют всем страницам сайта выглядеть единообразно.

Для связи с файлом, в котором описаны стили, используется тег *<link>*, расположенный в контейнере **<*head*>...</*head*>**.

В этом теге должны быть заданы два атрибута: *rel = "stylesheet"* и *href*, определяющий адрес файла стилей:

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

В работе предлагается, используя *html5* и *CSS3*, разработать веб-страницу формирования меню, содержащего главную ленту с графическим элементом и два пункта с выпадающими подменю (рисунок 2.1). Пример решения задачи приведен на листингах 2.1 и 2.2.

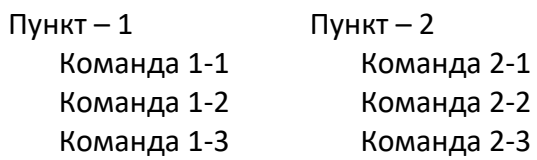

Рисунок 2.1 – Вид меню

Листинг 2.1 – *html*-код страницы

```
<!DOCTYPE html>
<html>
<head>
<title>Меню</title>
<link rel="stylesheet" type="text/css"
href="css/style.css" />
</head>
<body>
  <li><a href="">Пункт-1</a>
   <ul>
     <li><a href="#">Команда 1-1</a></li>
     <li><a href="#">Команда 1-2</a></li>
   </ul>
  </li>
 </ul>
</div>
```

```
<div id="mainmenu">
   <ul id="nav">
<li id="settings">
<a href="#"><img
src="images/settings.png"></a>
</li>
</body>
</html>
```

Листинг 2.2 – *CSS*-код страницы

```
#mainmenu      {
  float:left;     }
#mainmenu ul  {
  margin: 10;  /* отступ ленты меню сверху*/
  padding: 5;          /* отступ ленты меню
слева*/
  list-style: none;       }
#mainmenu ul li        {
  position: relative;
  float:left;     }
#mainmenu ul li ul, #mainmenu ul li ul li     {
  width:130px;
  color:red;    /*      */                  }
#mainmenu li ul        {
  position: absolute;
  left: 0;
  top: 29px;
  display: none;
  float:left;     }
#mainmenu ul li a      {
  float:left;
  color:blue;   /*Цвет текста глав меню  */
  width:80px; /*Длина ленты глав меню  */
  font-size:16px;
  padding: 10px 0 10px 0;
  text-align:center;
  background: #00f0f0;/*цвет фона гл  меню*/
}
#mainmenu li ul li a          {
  padding:5px 0 3px 10px;
  text-align:left;
  font-size:12px;
  width:80px;
  background: #EEEEEE;    /*Цвет          поля
группы выбр-го подменю*/     }
#mainmenu li ul li a:hover    {
  background: #0ffff0;   /*Цвет фона подменю
*/
  color:blue;  }
#settings a {            /*Графич-й эл-т */
  padding: 18px;
  height: 19px;         /*Высота графич эл-та */
  font-size: 10px;
  line-height: 24px;   }
* html #mainmenu ul li {float: left; height: 1%;}
* html #mainmenu ul li a { height: 1%; }
#mainmenu li:hover ul,
#mainmenu  li.over ul { display: block;       }
```

Порядок выполнения работы.

Выполнить задания, приведенные в колонках 2, 3, 4 и 5 таблицы 2.1. В столбце 2 указаны темы, для которых необходимо разработать веб-страницу формирования меню, в столбце 3 – параметры главной ленты меню: направление главной ленты меню (гориз – горизонтальное, верт – вертикальное); фон – цвет фона ленты меню; текст – цвет текста пунктов меню. В столбцах 4 и 5 указано количество пунктов второго и третьего подменю (два или три), цвет фона подменю и цвет текста его пунктов.

Главное меню должно отстоять от верхней границы окна браузера на 20 px, слева – на 12 px и содержать слева графический элемент.

Таблица 2.1 – Варианты заданий

Номер варианта	Тема	Главная лента меню: фон/текст	Подменю 1: пунктов/фон/текст	Подменю 2: пунктов/фон/текст
1	2	3	4	5
1	Торговая фирма	Гориз/Pink/Red	3/Blue/Blue	2/Aqua/Violet

Окончание таблицы 2.1

1	2	3	4	5
2	Ремонт авто	Верт/Coral/Blue	2/Navy/Navy	3/Olive/Aqua
3	Расписание занятий	Гориз/Gold/Navy	3/Olive/Aqua	2/Navy/Navy
4	Летняя одежда	Верт/Khaki/Aqua	2/Aqua/Violet	3/Blue/Blue
5	Ремонт компьютеров	Гориз/Plum/Black	3/Black/Red	2/Tan/Gold
6	Швейное ателье	Верт/Purple/Violet	2/Tan/Gold	3/Violet/Blue
7	Ремонт телевизоров	Гориз/Indigo/Tan	3/Violet/Blue	2/Purple/Navy
8	Головные уборы	Верт/Peru/Coral	2/Purple/Navy	3/Blue/Red
9	Зимняя одежда	Гориз/Gray/Gold	3/Blue/Red	2/Olive/Aqua
10	Продукты питания	Верт/Red/Plum	2/Olive/Aqua	3/Peru/ Gold
11	Детские игрушки	Гориз/Lime/Peru	3/Peru/ Gold	2/Pink/ Blue
12	Модели авто	Верт/Aqua/Gray	2/Pink/ Blue	3/Blue/Blue

Контрольные вопросы

1 Что такое таблица стилей?
2 Как подключить таблицу стилей к *html*-документу?
3 Что такое *CSS*-правило?
4 Какова структура правила *CSS*?
5 Какова структура определения *CSS*?
6 Что такое селектор в правиле *CSS*?
7 Из каких разделов состоит *html*-документ?

3 Лабораторная работа № 3. Создание сайта, содержащего сценарии на языке JavaScript

Скрипт (сценарий) – это последовательность действий, описанных с помощью скриптовых языков программирования *JavaScript*, *PHP*, *Perl*, *Python* и др. для расширения функциональных возможностей веб-страниц.

JavaScript является клиентским языком программирования, скрипты которого выполняются в браузере на локальном компьютере пользователя. Этот язык позволяет создавать веб-приложения и динамические сайты, способные взаимодействовать с пользователем.

Код *JavaScript* может содержаться непосредственно в *html*-документе либо в отдельном файле с расширением *.js*.

Для подключения *JavaScript*-кода (сценария) к *html*-документу используется тег <script>. При этом можно указать язык с помощью атрибута *type* (*type* = "*text/javascript*"). Однако этот атрибут можно опустить, т. к. в *JavaScript* значение атрибута *type* по умолчанию.

Скрипты включаются в *html*-документы несколькими способами:

– в теговом контейнере <*body*>...</*body*>:

<*body*>
…
<*script* > команды скрипта</*script*>
</*body*>

– в контейнере <*head*>...</*head*>, если скрипт представляет собой функцию, вызываемую в ответ на какое-либо событие:

<*head*>
…
<*script type=”text/javascript”*> команды сценария </script>
</*head*>

– во внешнем файле с расширением *js*:

<*head*>
…
<*script type = "text/javascript" src = "my.js"*> </*script*>
</*head*>

Порядок выполнения работы.

1 Создать строку текста из 25 первых букв русского алфавита: var str = 'abcde … '. Используя функцию *alert*, вывести символы с указанными в столбце «Задание 1 – номера символов» таблицы 2.1 номерами и разделить их номером варианта, например, для варианта 10: a10c10e10 и так далее.

2 Построить строку из цифр «Номера символов» первого задания. Из полученной строки, состоящей из 12 цифр, выделить четыре трехзначных числа. Используя полученные числа и операции, заданные в колонке «Задание 2 – операции» таблицы 2.1, построить оператор присваивания с полученным арифметическим выражением, результат вывести в консоль с помощью метода *console.log*.

3 Построить строку текста из букв, номера которых заданы в «Задание 1». Используя свойство *innerHTML* метода *document.getElementById(id),* вывести на страницу *html* построенную строку.

4 С помощью функции *confirm* построить запрос, содержащий две кнопки: Да и Нет. В зависимости от выбранной кнопки вычислить заданные в «Задании 3 – арифметические операции» таблицы 3.1 выражения: для Да – s, для Нет – n. Результат вывести в окно браузера.

Таблица 3.1 – Задания к лабораторной работе

Номер варианта	Задание 1 – номера символов	Задание 2 – операции			Задание 3 – арифметическое выражение	
					y =	f =
1	2	3			4	5
1	10, 13, 17, 19, 22, 25	+	–	*	$(a*c + b)^2/c$;	d/(f – e/2.7)
2	02, 05, 07, 09, 12, 14	+	–	/	$(a * b/c) – c^2$;	2*d/(f + e/4)

Окончание таблицы 3.1

1	2	3	4	5	6	7
3	04, 05, 07, 09, 12, 18	+	–	**%**	$(a – b*c)^2$;	d*(2*f – e/2)
4	13, 15, 17, 19, 21, 23	/	*	–	$(a^2$ /c+ b)/c;	d/(f/2.5 + e^2)
5	01, 04, 08, 12, 17, 21	/	*	+	$(a/c – b/c)^2/c$;	2*d/(f/1.4 – e/2.4)
6	03, 05, 06, 17, 18, 20	/	*	**%**	$(a + b/c^2)/c$;	d/(f*e/2.2) + d
7	05, 07, 09, 13, 16, 18	*	+	–	$(a – b/2.4)/c^2$;	d/(f*e + 2.7*d) – d
8	05, 07, 09, 12, 14, 18	*	+	/	$(0.72 + b/a^2)/2*c$;	d/(f*e – 2.5*d) + e
9	02, 09, 16, 17, 18, 21	*	+	**%**	$(a/c – b/a*2)/c^2$;	d/(e*f/2.1) + d*e/f
10	08, 12, 18, 19, 20, 24	–	*	/	$(a + b/c^2)/c-a$;	d/(f*e/2.8 + d) – 2*f
11	01, 03, 05, 07, 10, 11	–	*	+	$(a – b/c/2)/c^a$;	d/(f*e + 2.3*f) + d
12	16, 18, 19, 21, 23, 25	–	*	**%**	(a + b/c+2)/c–2*b;	d/(f*e – 2.5/f) – d

Контрольные вопросы

1 Как встроить *JavaScript*-код в *html*-документ?
2 Какие комментарии используются в языке *JavaScript*?
3 Как выделить символ из строки текста?
4 Какие функции вывода на страницу *html* Вы знаете?
5 Прокомментируйте технологию использования метода *document.getElementById(id)* вывода на страницу *html*.
6 Какие математические операции используются в *JavaScripn*?
7 Как извлечь символ строки по его номеру?
8 Для каких целей используются методы *prompt* и *Confirm*?
9 Для чего используются методы *document.write()*, *alert()*, *console.log()*?
10 Какие способы включения *JavaScript*-кода в *html*-документ Вы знаете?

4 Лабораторная работа № 4. Изучение операторов ветвлений и циклов JavaScript

Для организации ветвлений в программах на языке *JavaScript* используются конструкции *if*, *else*, *switch*, обеспечивающие выполнение определенной команды или набора команд только при условии истинности некоторого логического выражения.

Цикл представляет собой разновидность управляющей конструкции, предназначенной для организации многократного выполнения набора инструкций.

Синтаксис конструкции *if*

```
if (логическое выражение) {
        код если логическое выражение = true
} else {
        код если логическое выражение = false
}
```

Схема оператора ветвления *if else*

```
if (условие) {
        код запустится, если условие вернёт true
} else {
        код запустится, если условие вернёт false
}
```

Например:

```
var n1 = 5;
var n2 = 3;
if (n1 > n2) {
        alert("Условие возвратило true");
} else { alert("Условие возвратило false"); }
```

Синтаксис конструкции *switch*

```
switch (переменная) {
  case '1':
     код выполняемый, если переменная имеет значение 1;
  break;
  ...
  case 'n':
     код, выполняемый, если переменная имеет значение n;
  break;
  default:
     код, выполняемый, если переменная не совпала ни с одним значением;
  break;
}
```

Синтаксис конструкции *for*

```
for (начальное значение; условие окончания цикла; команды после прохода цикла) {
     тело цикла
}
```

Примеры конструкции for

1 Вывести четные числа от 20 до 0:

```
for (var i = 20; i >= 2; i -= 2) {
        document.write(i + ' ');
}
```

2 Вывести элементы массива:

```
var books = ['JavaScript', 'PHP', 'html', 'CSS'];
for (var i = 0; i < books.length; i++) {
        var str = (i + 1) + '. ' + books[i] + '<br>';
        document.write(str);  }
```

Синтаксис конструкции *while*

```
while ( пока выражение истинно ) {
     выполнять код      }
```

Пример конструкции while – Вывести числа от 1 до 10:

```
var i = 1;
while (i <= 10) {
    document.write(i + ' ');
    i++;          }
```

Инструкция *break* используется для принудительного выхода из цикла, а инструкция *continue* – для принудительного перехода к следующей итерации цикла.

Пример инструкции break – Вывести четные числа от 2 до 20:

```
var result = ' ';
for (var i = 2; i <= 20; i++) {
    if (i % 2) continue;
    result += i + ' ';    }
document.write(result);
```

Порядок выполнения работы.

Разработать консольное приложение на языке *JavaScript* для решения следующих задач.

1 Ввести переменную *lang*, которая может принимать значения: рус, анг, бел или нем. В переменной *msw* сформировать массив дней недели на русском, английском, белорусском или немецком языке в зависимости от варианта. Задачу решить с помощью оператора *if*; *switch-case* или многомерного массива.

2 Дана строка вида 'a12cdef345'. Проверить, является ли символ с заданным номером *k* этой строки буквой, а сумма ее цифр – четной.

3 Ввести дату и по ней определить: ВГ – время года (зима, весна, лето, осень); ДМ – декаду месяца; МГ – месяц; ВсГ – високосный/не високосный год.

4 Ввести массив из 25 целых вещественных чисел и определить: СЧЭ – сумму четных элементов; ПЭНН – произведение элементов с нечетными номерами; СЭКЗ – сумму элементов, номера которых кратны четырем; ПНЧЭ – произведение нечетных элементов массива.

Варианты заданий приведены в таблице 4.1.

Таблица 4.1 – Варианты заданий

Номер варианта	Задача 1		Задача 2		Задача 3	Задача 4
	lang	операт/ массив	строка	k	определить	определить
1	2	3	4	5	6	7
1	анг	If	'5abcde12345'	11	ВГ; ДМ	СЧЭ; ПЭНН
2	рус	switch-case	'1ab123cde45'	7	ВГ; МГ	СЧЭ; СЭКЗ
3	бел	массив	'4abcd12345e'	2	ВГ; ВсГ	СЧЭ; ПНЧЭ
4	анг	If	'3ab12cde345'	5	ДМ; ВГ	ПЭНН; СЧЭ
5	рус	switch-case	'7a12bcde345'	4	ДМ; МГ	ПЭНН; СЭКЗ
6	бел	массив	'4a1234bcde5'	9	ДМ; ВсГ	ПЭНН; ПНЧЭ
7	анг	If	'r123abc45de'	3	МГ; ВГ	СЭКЗ; СЧЭ

Окончание таблицы 4.1

1	2	3	4	5	6	7
8	бел	switch-case	's123abcde45'	5	МГ; ДМ	СЭК3; ПЭНН
9	рус	массив	'r12ab34cde5'	7	МГ; ВсГ	СЭК3; ПНЧЭ
10	нем	If	'k123abc45de'	6	ВсГ; ВГ	ПНЧЭ; СЧЭ
11	анг	switch-case	'1ab23c45de'	9	ВсГ; ДМ	ПННЭ; ПЭНН
12	рус	массив	'1ab2345cde'	10	ВсГ; МГ	ПННЭ; СЭК3

Контрольные вопросы

1 Какие варианты использования оператора *if* Вы знаете?
2 Прокомментируйте назначение и структуру оператора *switch-case*.
3 Прокомментируйте синтаксис оператора *for*.
4 Для чего предназначена инструкция *break*?
5 Какие операторы цикла Вы знаете?
6 Для чего предназначен оператор *for in*?
7 Для чего используется инструкция *continue*?

5 Лабораторная работа № 5. Изучение функций и методов JavaScript

Все значения в *JavaScript*, за исключением *null* и *undefined*, содержат набор вспомогательных функций и значений, доступных через точку. Такие функции называют методами, а значения – свойствами. Чтобы вызвать метод объекта, используется следующий синтаксис: *имяОбъекта.имяМетода()*.

В *JavaScript* функция является значением, поэтому её можно присваивать переменным, элементам массива, свойствам объектов, передавать в качестве аргумента функциям и возвращать в качестве значения из функций.

Методы *JavaScript* – это действия, которые могут выполняться над объектами, в то же время это свойства, содержащие определение функции, например,

```
var person = {
   firstName: "John",
   lastName : "Doe",
   id : 5566,
   fullName : function() {
      return this.firstName + " " + this.lastName;
   }
};
```

Ключевое слово *this* в данном примере относится к владельцу функции, т. е. к объекту *Person*, который владеет функцией *function()*.

Методом объекта *JavaScript* может быть только функция, а значением свойства объекта – любой тип данных, за исключением функции.

Основные методы *JavaScript*

Методы строк. Строка в *JavaScript* является одновременно и объектом *string*, и переменной, поэтому может быть создана двумя способами:

```
st1 = new String("Строка – это объект")
st2 = "Строка – это переменная"
```

В *JavaScript* используются следующие методы строк:

charAt() – извлекает из строки символ, находящийся в указанной позиции;

charCodeAt() – возвращает код юникода символа, находящегося в указанной позиции (16-разрядное целое число между 0 и 65 535);

concat() – выполняет конкатенацию одного или нескольких значений со строкой, преобразует все аргументы в строки и добавляет их в конец строки;

indexOf (подстрока, начало) – выполняет поиск в строке от начала к концу;

lastIndexOf() – выполняет поиск символа или подстроки в строке с конца;

match() – выполняет сопоставление строк по шаблону с помощью регулярного выражения.

Для выделения нескольких символов строки используется метод *substr()*.

Замена подстроки текста выполняется методом *replace()*, построенным на использовании регулярных выражений.

Методы вывода:

alert – выводит модальное окно с сообщением;

confirm – выводит сообщение в окне с двумя кнопками: «ОК» и «ОТМЕНА» – и возвращает выбор посетителя;

prompt – выводит окно с указанным текстом и полем для пользовательского ввода;

setInterval – выполняет код или функцию через указанный интервал времени.

Метод *document.write()* выводит на страницу переданные ему аргументы.

Глобальные методы *JavaScript*:

alert – выводит модальное окно с сообщением;

clearInterval – останавливает выполнение кода, заданное *setInterval*;

clearTimeout – отменяет выполнение кода, заданное *setTimeout*;

confirm – выводит сообщение в окне с двумя кнопками: «ОК» и «ОТМЕНА» – и возвращает выбор посетителя;

decodeURI – раскодирует *URI*, закодированный при помощи *encodeURI*;

decodeURIComponent – раскодирует *URI*, закодированный при помощи *encodeURIComponent*;

encodeURI – кодирует *URI*, заменяя каждое вхождение определенных символов на *escape*-последовательности, представляющие символы в кодировке *UTF*-8;

encodeURIComponent – кодирует компоненту *URI*, заменяя определенные символы на соответствующие *UTF*-8 *escape*-последовательности;

eval – выполняет строку *javascript*-кода без привязки к конкретному объекту;

isFinite – проверяет, является ли аргумент конечным числом;
isNaN – проверяет, является ли аргумент NaN.

Порядок выполнения работы.

Выполнить задания.

1 Заполнить массив из 14 элементов вещественными числами, используя заданную в столбце 2 таблицы 5.1 функцию. Рассортировать массив в указанном в столбце 3 порядке.

2 Вывести текущую дату в заданном в столбце 4 формате и определить:

– используя функцию *DataParse*, количество миллисекунд, прошедших с 01.01.1970 г. по текущий момент;

– используя метод *getTime*, количество секунд от 01.01.1970 г. по текущий момент;

– используя метод *getDay*, номер дня недели и название дня Вашего рождения.

3 Вычислить значение элемента, заданного в столбце 5.

Таблица 5.1 – Варианты заданий

Номер варианта	Задание 1		Задание 2	Задание 3
	Функция	Сортировать	Даты	Вычислить
1	2	3	4	5
1	sin(x)	По возрастанию	Год, месяц, день	Площадь усеченного конуса
2	cos(x)	По убыванию	Месяц, день, час	Площадь трапеции
3	tg(x)	По возрастанию	Час, минута, секунда	Площадь параллелограмма
4	ctg(x)	По убыванию	День, час, минута	Максимальное значение среди трех
5	sin(x/2)	По возрастанию	Год, месяц, день	Максимальное значение среди пяти
6	cos(x/2)	По убыванию	Месяц, день, час	Объем конуса
7	tg(x/2)	По возрастанию	Час, минута, секунда	Объем усеченного конуса
8	ctg(x/2)	По убыванию	День, час, минута	Объем пирамиды
9	$\sin(x^2)$	По возрастанию	Год, мес, день	Объем усеченной пирамиды
10	$\cos(x^2)$	По убыванию	Месяц, день, час	Площадь шестиугольника
11	$tg(x^2)$	По возрастанию	Час, минута, секунда	Площадь кольца
12	$ctg(x^2)$	По убыванию	День, час, минута	Площадь цилиндра

Контрольные вопросы

1 Для чего используются методы *alert()*, *prompt()* и *document.write()*?
2 Какие методы объекта *Math* Вы знаете?
3 Прокомментируйте назначение метода *document.getElementById(id)*.
4 Какие два способа создания строки Вы знаете?
5 Что понимается под методом в *JavaScript*?
6 Какие методы вывода в *JavaScript* Вы знаете?
7 Через какой объект выполняется работа с датами в *JavaScript*?

6 Лабораторная работа № 6. Изучение приемов работы с массивами JavaScript

Элементы одного и того же массива в языке *JavaScript* могут иметь разные типы, т. е. являются нетипизированными, и быть объектами или другими массивами, что позволяет создавать сложные структуры данных, такие как массивы объектов и массивы массивов.

Отсчет индексов массивов в языке *JavaScript* начинается с нуля, и для них используются 32-битные целые числа. Массивы в *JavaScript* являются динамическими: они могут увеличиваться и уменьшаться в размерах по мере необходимости, поэтому можно не указывать фиксированные размеры массивов при их создании или повторном распределении памяти при изменении их размеров.

Создать массив проще всего с помощью литерала, который представляет собой простой список разделенных запятыми элементов в квадратных скобках. Значениями литерала массива могут быть константы, выражения, литералы объектов:

```
var empty = [];                 // Пустой массив
var numbers = [2, 3, 5, 7, 11];// Массив с пятью числовыми элементами
var misc = [ 1.1, true, "a", ];   // 3 элемента разных типов + завершающая запятая
var base = 1024;
var table = [base, base+1, base+2, base+3];// Массив с переменными
var arrObj = [[1,{x:1, y:2}], [2, {x:3, y:4}]];// 2 массива внутри, содержащие объекты
```

Синтаксис литералов массивов позволяет вставлять необязательную завершающую запятую, т. е. литерал [,,] соответствует массиву с двумя элементами, а не с тремя.

Другой способ создания массива – конструктор *Array()*, который можно вызвать тремя разными способами: без аргументов; с единственным числовым аргументом, определяющим длину массива; с явным указанием значений первых двух или более элементов или одного нечислового элемента:

```
var arr = new Array();                  // пустой массив, эквивалентный литералу []
var arr = new Array(10);                // пустой массив указанной длины
var arr = new Array(4, 3, 5, 2, 1, "тест");   //массив с явным указанием элементов
```

Массив является специализированной разновидностью объекта, поэтому квадратные скобки, используемые для доступа к его элементам, действуют аналогично доступу к свойствам объекта. Интерпретатор *JavaScript* преобразует указанные в скобках числовые индексы в строки, например, индекс 1 – в строку "1", а затем использует эти строки как имена свойств.

Следует четко отличать индексы в массиве от имен свойств объектов. Все индексы массива являются именами свойств, но только свойства с именами, представленными целыми числами, являются индексами. Массивы являются объектами, и к ним можно добавлять свойства с любыми именами. Однако если

затрагиваются свойства, которые являются индексами массива, то массивы реагируют на это, обновляя при необходимости значение свойства *length*.

В качестве индексов массивов допускается использовать отрицательные и нецелые числа. В этом случае числа преобразуются в строки, которые используются как имена свойств.

Добавление и удаление элементов массива. Самый простой способ добавления элементов массива – это присваивание значений его новым индексам. Для добавления одного или более элементов в конец массива можно также использовать метод *push()*:

```
var arr = [];                 // Создать пустой массив
arr.push('zero');             // Добавить значение в конец массива
arr.push('one',2);            // Добавить еще два значения в массив
```

Добавить элемент в конец массива можно также, присвоив значение элементу *arr[arr.length]*. Для вставки элемента в начало массива можно использовать метод *unshift()*, при этом существующие элементы в массиве смещаются в позиции с более высокими индексами.

Удалять элементы массива можно как обычные свойства объекта – с помощью оператора *delete*:

```
var arr = [1,2,'three'];
delete arr[2];
2 in arr;             // false, индекс 2 в массиве не определен
arr.length;           // 3: оператор delete не изменяет свойство length массива
```

Многомерные массивы. *JavaScript* не поддерживает настоящие многомерные массивы, но позволяет имитировать их при помощи массива из массивов. Для доступа к элементу данных в массиве массивов достаточно дважды использовать оператор [ ].

Методы класса *Array*

Array.join() – преобразует все элементы массива в строки, объединяет их и возвращает получившуюся строку.

Array.reverse() – меняет порядок следования элементов в массиве на обратный и возвращает переупорядоченный массив.

Array.sort() – сортирует элементы в исходном массиве и возвращает отсортированный массив.

Array.concat() – создает и возвращает новый массив, содержащий элементы исходного массива, для которого был вызван метод *concat()*, и значения всех переданных ему аргументов.

Array.slice() – возвращает фрагмент, или подмассив указанного массива.

В языке *JavaScript* используются также ассоциативные массивы, в которых вместо индексов, применяются ключи, а для их объявления используется оператор *var*, после которого прописывается его имя, знак «равно» и в фигурных скобках – свойства и значения.

Порядок выполнения работы.

Выполнить задания, варианты которых приведены в таблице 6.1.

1 Создать одномерный массив R из k элементов, указанных в столбце 2 таблицы 1.6. Добавить m числовых элементов в его начало и n текстовых – в конец.

2 Создать двухмерный массив размерностью m × n (столбец 4 таблицы 1.6) и заполнить случайными равномерно распределенными числами. Строки с четными номерами рассортировать.

3 Создать двухмерный массив размерностью k × m, содержащий текстовые и числовые элементы. Используя метод *Array.join()*, преобразовать все числовые элементы массива в строки; изменить порядок следования элементов массива на обратный.

4 Создать двухмерный текстовый массив размерностью m × k. Используя методы *unshift()* и *push()*, добавить k - 3 строк в начало массива и с помощью методов *shift()* и *pop()* удалить m - 2 строк из конца заданного массива.

Таблица 6.1 – Варианты заданий

Номер варианта	Задание 1		Задание 2	Задание 3		Задание 4
	строки	k, m, n	m, n	сортировать	k, m	m, k
1	2	3	4	5	6	7
1	['a', 'b', 'c']	3, 2, 4	4, 6	По возрастанию	7, 8	6, 9
2	[1, 2, 3]	4, 3, 2	5, 8	По убыванию	6, 5	7, 6
3	['p', 'n', 'k']	2, 1, 5	3, 5	По возрастанию	8, 6	8, 9
4	[7, 5, 8]	4, 2, 3	4, 6	По убыванию	5, 8	5,7
5	['r', 's', 't']	3, 4, 4	5, 7	По возрастанию	6, 7	5, 8
6	[2, 6, 8]	5, 3, 2	4, 6	По убыванию	5, 8	5, 9
7	['k', 'm', 's']	4, 1, 3	5, 8	По возрастанию	8, 7	6, 5
8	[3, 7, 5]	2, 5,4	3, 6	По убыванию	6, 8	6, 7
9	['q', 'z', 's']	5, 1, 3	4, 5	По возрастанию	5, 6	6, 8
10	[2, 9, 4]	3, 4, 5	5, 7	По убыванию	8, 8	6, 6
11	['t', 'r', 'n']	4, 2, 3	2, 6	По возрастанию	7, 7	7, 6
12	[4, 7, 2]	3, 5,2	3, 8	По убыванию	6, 5	7, 7

Контрольные вопросы

1 Какие методы создания массивов Вы знаете?
2 Что такое динамический массив в *JavaScript*?
3 Что может быть элементом массива в *JavaScript*?
4 С какого значения ведется отсчет индексов массивов в *JavaScript*?
5 Как построить доступ к элементу массива?
6 Какой тип может иметь индекс элемента массива?
7 Как понимать «массив – специализированная разновидность объекта»?

7 Лабораторная работа № 7. Изучение приемов работы с элементами управления JavaScript

Язык *JavaScript* позволяет создавать сложные веб-элементы управления сайтами, среди которых сложные меню, специализированные деревья и сложные сетки, а также два специальных – генератор всплывающих окон и динамически меняющаяся кнопка.

Один из способов, позволяющих показать пользователю дополнительный контент, – это всплывающие (*popup*-) окна.

В недавнем прошлом всплывающими окнами злоупотребляли многие сайты, нацеленные на показ рекламы, и загружали пользователей множеством объявлений. Поэтому современные браузеры блокируют всплывающие окна.

Всплывающее окно достаточно просто отображается с помощью функции *window.open()* в блоке *JavaScript*:

```
window.open('http://www.google.com', 'myWindow',
   'toolbar=0, height=500, width=800, resizable=1, scrollbars=1');
window.focus();
```

Функция *window.open()* принимает параметры: ссылка на новую страницу и имя фрейма окна, в которое позже должен быть загружен новый документ посредством другой ссылки. Третий параметр – разделенная запятыми строка, конфигурирующая стиль и размер всплывающего окна с помощью атрибутов:

– *height* – высота и *width* – ширина в пикселях;

– *toolbar* – панель инструментов и *menuBar* – строка меню, которые могут быть установлены в 1 или 0 в зависимости от того, требуется ли отображение этих элементов;

– *resizable* = 1 – рамка изменяемого размера, = 0 – фиксированного;

– *scrollbars* = 1, если требуются линейки прокрутки, = 0 – если нет.

Чтобы закрыть *popup*-окно, необходимо вызвать функцию *newWindow.close()*. Метод *close()* можно вызвать для любого объекта *window*, но *window.close()* игнорируется почти всеми браузерами, если окно было открыто не с помощью *window.open()*.

Эффективным элементом управления в *JavaScript* является динамически меняющаяся кнопка, которая выводит на экран одно изображение, если она появляется на веб-странице впервые, при задержке над ней курсора мыши – другое, при щелчке на этой кнопке третье.

Для обеспечения такого эффекта кнопка обычно состоит из дескриптора <*img*>, который обрабатывает *JavaScript*-события *onclick*, *onmouseover* и *onmouseout*. Эти события вызывают функции, меняющие изображения для текущей кнопки:

```
function swapImg(id, url) {
   var elm = document.getElementById(id);
   elm.src = url;
}
```

В этом случае сконфигурированный дескриптор <img> выглядел бы следующим образом, *html*:

```
<img id="img" src="activebtn.png" style="cursor:pointer; margin:120px"
   onmouseover="swapImg('img', 'hoverbtn.png')"
   onmouseout="swapImg('img', 'activebtn.png')" />
```

Порядок выполнения работы.

Выполнить задания, варианты которых приведены в таблице 7.1.

1 Написать сценарий, позволяющий продемонстрировать изменения размеров и положения горизонтальной линии на странице *html*.

2 Написать сценарий формирования анкеты данных сотрудника, указанных в столбце 3 таблицы 7.1.

3 Написать сценарий обработки анкеты слушателя курсов повышения квалификации, содержащей: курс (первый, второй, третий); язык общения с преподавателем; изучаемые дисциплины; продолжительность курса; форму образования (очная, заочная, дистанционная); изучаемые дисциплины; стоимость. В зависимости от этих параметров определяется стоимость отдельного курса и стоимость всего обучения. Выбор значений выполнить с помощью флажков и выпадающего меню.

Таблица 7.1 – Варианты заданий

Номер варианта	Задание 1: длина линии, %; толщина, пикс	Задание 2 (данные сотрудника): МР – место рождения; НАЦ – национальность; ОБР – образование; ДР – дата рождения; СП – семейное положение	Задание 3 (поля выбора): ПР – продолжительность; ФО – форма образования; ФИОП – ФИО преподавателя; ИД – изучаемые дисциплины
1	45 %; 2 пикс	Пол, МР, ОБР	Курс, язык, ПР, стоимость
2	72 %, 3 пикс	ДР. пол, должность.	Курс, язык, ФО,
3	85 %, 4 пикс	МР, возраст, пол	Курс, язык, ФИОП,
4	57 %, 2 пикс	МР, должность, пол	Курс, язык, ПР, ИД
5	80 %, 3 пикс	Пол, должность, ДР	Курс, язык, ФО, стоимость
6	90 %, 2 пикс	НАЦ, пол, возраст	Курс, язык, ФИОП, ИД
7	50 %, 2 пикс	СП, пол, должность	Курс, язык, ФО, стоимость
8	70 %, 3 пикс	Пол, НАЦ, возраст	Курс, язык, ПР, ИД
9	80 %, 4 пикс	МР, возраст, должность	Курс, язык, ФИОП,
10	35 %, 2 пикс	ДР, должность, возраст	Курс, язык, ПР, ИД
11	45 %, 3 пикс	НАЦ, должность, образование	Курс, язык, ФО, стоимость
12	75 %, 4 пикс	МР, возраст, должность	Курс, язык, ФИОП, ИД

Контрольные вопросы

1 Для чего используется генератор всплывающих (*popup*-) окон?
2 Что такое динамически меняющаяся кнопка?
3 Для чего используется функция *window.open()*?

4 Какие параметры содержит функция *window.open()*?
5 Как закрыть *popup*-окно?
6 Где используется и для чего свойство *z-index*?
7 Как работают динамически изменяющиеся кнопки?

8 Лабораторная работа № 8. Изучение основных методов JQuery

Методы *jQuery* позволяют манипулировать содержимым веб-страницы. Они присваивают элементам, отобранным в *jQuery*-объектах, заданные действия. В результате этого происходит динамическое изменение элементов и их содержимого.

Каждый метод *jQuery* либо сам что-либо возвращает, либо получает параметр и выполняет указанные в параметре действия.

В общем виде синтаксис для вызова метода *jQuery* имеет следующий вид:

$("селектор").имяМетода(параметры);

Динамическое изменение элементов веб-страниц. Библиотека *jQuery* упрощает процесс отбора элементов *html*-страниц. С помощью методов *jQuery* производятся манипуляции с объектной моделью документа DOM. Чтобы отобрать группу элементов, нужно передать селектор функции *jQuery*. В качестве селектора элемента может выступать сам элемент, его идентификатор или класс, а также комбинация селекторов: *$("a"); $("#some-id"); $(".someclass"); $("header > ul:has(a)")*.

Функция *$()* возвращает объект *jQuery*, содержащий массив элементов DOM – так называемый обернутый набор, соответствующий указанному селектору. Большинство методов по завершении действий возвращает первоначальный набор элементов.

JQuery – это одна из наиболее известных библиотек, написанная на языке *JavaScript* для упрощения программирования веб-страниц. Это файл с расширением .js, который подключается к веб-странице как фрагмент скрипта и загружается в браузер вместе с веб-страницей.

Чтобы включить *JQuery* в веб-страницу, достаточно скачать последнюю версию библиотеки, например, файл *jquery*-1.9.1.js с сайта *jquery.com*, положить его в ту же папку, где лежит текст веб-страницы, а в текст веб-страницы вставить *<script src="jquery-1.9.1.js"></script>*.

Файл *jquery-1.9.1.js* при этом имеет объем более 200 Кбайт, что может замедлять загрузку веб-страницы в браузере пользователя.

Если веб-страница маленькая и важно, чтобы все «летало» и загрузка библиотеки *JQuery* ничего не замедляла, то существует альтернативный метод загрузки *JQuery* – с сайта *Google*:

<script src="//ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js">
</script>

Второе важное характерное для *jQuery* применение состоит в создании *AJAX*-элементов, т. е. тех элементов страницы, которые отсылают на сервер данные и получают ответ без перезагрузки страницы. К таким элементам можно отнести форму «Управление корзиной для интернет-магазина», пагинацию (нумерацию страниц сайта), вывод информера погоды и многое другое.

Порядок выполнения работы.

Используя библиотеку *Jquery*, создать сайт. Варианты заданий указаны в таблице 8.1.

Таблица 8.1. – Варианты заданий

Номер варианта	Используя библиотеку *Jquery*, создать сайт, содержащий
1	Выпадающее меню
2	Плавающее меню
3	Фотогалерею
4	Всплывающие окна
5	Слайдеры (блоками на странице, в пределах которых с установленной периодичностью происходит демонстрация анонсов новостей, статей или изображений)
6	Перемещающиеся блоки
7	Изменение прозрачности элементов
8	Подсвечивание текста
9	Переливание цвета текста разными оттенками
10	Плавающее меню
11	Всплывающие окна
12	Изменение прозрачности элементов

Контрольные вопросы

1 Что такое *JQuery*?
2 Что такое слайдер?
3 Какие действия выполняют методы *JQuery*?
4 Прокомментируйте синтаксис вызова методов *jQuery*.
5 Что понимается под селектором *jQuery*?
6 Какие основные правила использования *jQuery* Вы знаете?
7 Как подключить библиотеку *jQuery* к веб-странице?

9 Лабораторная работа № 9. Изучение основных событий JQuery

События *jQuery*, представляющие собой момент, в который что-либо происходит, например щелчок кнопки мыши, помогают сделать веб-страницы интерактивными, реагирующими на простейшие действия пользователя.

Момент, в который произошло событие, называется запуском события. События могут срабатывать при выполнении различных операций с веб-страницей. Помимо этого, и сам браузер может стать источником событий.

Управление веб-страницей производится с помощью следующих событий: мыши; документа/окна; форм; клавиатуры; *jQuery*.

События мыши:

.click() – запускается при нажатии и отпускании кнопки мыши, применяется к ссылкам, картинкам, кнопкам, абзацам, блокам и т. д.;

.dblclick() – запускается при двойном нажатии и отпускании кнопки мыши, например, при открытии какой-либо папки;

.mousedown() – происходит во время нажатия кнопки мыши, например, при перетаскивании элементов;

.mousemove() – запускается при перемещении указателя мыши по элементу.

События документа/окна:

.load() – запускается, когда браузер загрузит все файлы веб-страницы: *html*-файлы, внешние *css*- и *Javascript*-файлы, медиафайлы;

.resize() – запускается, когда пользователь изменяет размер окна браузера;

.scroll() – запускается, когда пользователь использует полосы прокрутки, либо прокручивает веб-страницу с помощью колесика мыши, либо использует для этих целей клавиши клавиатуры (*pgup, pgdn, home, end*);

.unload() – запускается, когда пользователь собирается покинуть страницу, щелкая по ссылке для перехода на другую страницу, закрывает вкладку страницы или окно браузера.

События форм:

.blur() – запускается, когда поле формы выводится из фокуса, например, при переходе в другое поле формы;

.change() – запускается при изменении статуса поля формы, например, при выборе пункта из выпадающего меню;

.focus() – запускается при переходе в поле формы, при щелчке на нем кнопкой мыши или клавишей табуляции;

.reset() – позволяет вернуть форму в первоначальное состояние, отменив сделанные изменения;

.select() – запускается при выделении текста внутри текстового поля формы;

.submit() – запускается при отправлении заполненной формы с помощью щелчка по кнопке «Отправить» или нажатии клавиши «Enter», когда курсор помещен в текстовом поле.

События клавиатуры:

.keydown() – запускается при нажатии клавиши перед событием *keypress*;

.keypress() – запускается при нажатии на клавишу до тех пор, пока клавиша не будет отпущена;

.keyup() – запускается при отпускании клавиши.

События *jQuery*:

.hover() – позволяет одновременно решить две задачи, связанные с событием наведения указателя мыши и событием снятия указателя мыши в отношении выбранного объекта;

.toggle() – работает аналогично событию *hover()* с разницей в том, что оно запускается от щелчка кнопкой мыши. Например, можно открыть выпадающее меню одним щелчком и скрыть вторым.

Объект события: при запуске события браузер сохраняет информацию о нём в объекте события, который содержит данные, собранные в момент, когда событие произошло. Обработка события происходит с помощью функции, при этом объект передается функции как аргумент–переменная *evt*.

Объект события имеет различные свойства, наиболее распространенные из которых следующие:

pageX – расстояние (px) от указателя мыши до левого края окна браузера;

pageY – расстояние (px) от указателя мыши до верхнего края окна браузера;

screen – расстояние (px) от указателя мыши до левого края монитора;

screenY – расстояние (px) от указателя мыши до верхнего края монитора;

shiftKey – TRUE, если была нажата клавиша «SHIFT», когда происходило событие;

which – используется для определения числового кода нажатой клавиши (вместе с *shiftKey*);

target – означает, что по объекту события щелкнули кнопкой мыши (например, для события *click()*);

data – объект, использованный с функцией *bind()* для передачи данных функции, управляющей событием.

Порядок выполнения работы.

Используя события, указанные в таблице 9.1, и библиотеку *Jquery*, создать сайт.

Таблица 9.1 – Варианты заданий

Номер варианта	Используя библиотеку *Jquery*, создать сайт, содержащий
1	События мыши .click() и .mousedown()
2	События мыши .dblclick() и .mousemove()
3	События документа/окна .load() и .scroll()
4	События документа/окна .resize() и .unload()
5	События форм. blur(), .focus() и .select()
6	События форм .change(), .reset() и .submit()

Окончание таблицы 9.1

Номер варианта	Используя библиотеку *Jquery*, создать сайт, содержащий
7	События клавиатуры .keydown() и .keyup()
8	События клавиатуры .keypress() и .keyup()
9	Событие jQuery .hover()
10	Событие jQuery .toggle()
11	События мыши .click() и .mousemove()
12	События форм. blur(), .focus() и .submit()

Контрольные вопросы

1 Что представляют собой события *jQuery*?
2 С помощью каких событий производится управление веб-страницей?
3 Какие события мыши Вы знаете?
4 Что такое объект события?
5 Какие события документа/окна Вы знаете?
6 Какие события клавиатуры Вы знаете?
7 Какие события форм Вы знаете?

10 Лабораторная работа № 10. Установка локального сервера MAMP

При разработке и отладке серверных приложений используются локальные сервера, наиболее популярным среди которых является веб-сервер *Apache*.

Подготовка, установка, настройка и работа с веб-сервером *Apache* вызывает некоторые сложности, поэтому чаще всего используют уже настроенные и надежно работающие в среде Windows системы управления контентом CMS (Content Management System), такие как DENVER,WAMP, MAMP и другие.

Связь внешней программы с веб-сервером выполняется с использованием CGI (Common Gateway Interface – общий интерфейс шлюза), а программу, позволяющую использовать консоль ввода и вывода для взаимодействия с клиентом и работающую по интерфейсу CGI, принято называть шлюзом, но используется также название *скрипт* (сценарий) или CGI-программа.

Для разработки серверных приложений на языке PHP используется интерпретатор PHP, который представляет собой либо внешнюю CGI-программу, либо динамическую библиотеку, которую необходимо подключить к веб-серверу, чтобы вместо кода PHP-скриптов клиенту выдавались результаты ее выполнения.

Традиционно совместно с веб-сервером *Apache* и интерпретатором PHP используется СУБД MySQL.

Самой популярной CMS, распространяемой по открытому лицензионному соглашению, является *WordPress*. По данным веб *Technology Surveys*, на этом

движке по состоянию на ноябрь 2018 г. разработано 32,3 % от общего числа существующих сайтов, а также 59,5 % сайтов, использующих CMS. С помощью *WordPress* можно создать интернет-магазин, личный блог, корпоративный сайт, информационный портал, отраслевой ресурс, галерею мультимедиа и др.

Принципы построения сайтов в среде *WordPress* понятны на интуитивном уровне. После создания и настройки сайта необходимо опубликовать контент, а чтобы сайт был эффективным, контент должен быть качественным и полезным для аудитории, поэтому его необходимо регулярно обновлять, что является самой сложной и ответственной работой.

Одним из наиболее простых в использовании и легко настраиваемых CMS является также MAMP, который доступен по адресу https://www.mamp.info/en/downloads/.

После скачивания MAMP необходимо кликнуть на иконку установочного пакета, чтобы запустить процесс распаковки и установки MAMP на компьютер. После всех успешных действий установки появится диалоговое окно локального сервера (рисунок 10.1).

Рисунок 10.1 – Диалоговое окно MAMP

Особого внимания здесь заслуживает ссылка с шестеренкой и надписью Preferences – Настройки и привилегии, активизация которой выводит окно с пятью вкладками (рисунок 10.2).

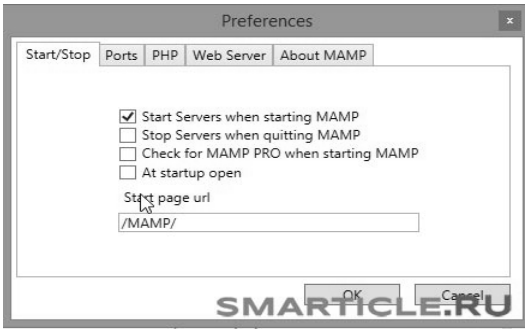

Рисунок 10.2 – Меню Preferences

После успешных настроек должны загореться два пункта зеленым цветом – ***Apache*** и MySql (рисунок 1), подтверждающих, что сервер работает.

Далее следует перейти на стартовую страницу, нажав на ссылку *Open Start page*, после чего должен открыться браузер, в адресной строке которого

появится локальный путь *localhost/MAMP*, по которому будет выполняться обращение к файлам сайта.

После этого следует перейти в навигационное меню, в котором необходимо отметить только один раздел Tools (Инструментарий), где расположена ссылка для доступа в *phpMyAdmin*.

Вторая важная вкладка необходима для разрешения конфликта между Скайпом – Ports (Порты): порт Апач – 80, MySql-порт – 3306.

Остальные вкладки можно не редактировать.

Далее, после проведенных настроек, можно запустить сервер, нажав на ссылку *Start Servers* или на ссылку *Open Start page*, перейти на стартовую страницу и открыть браузер (рисунок 10.3).

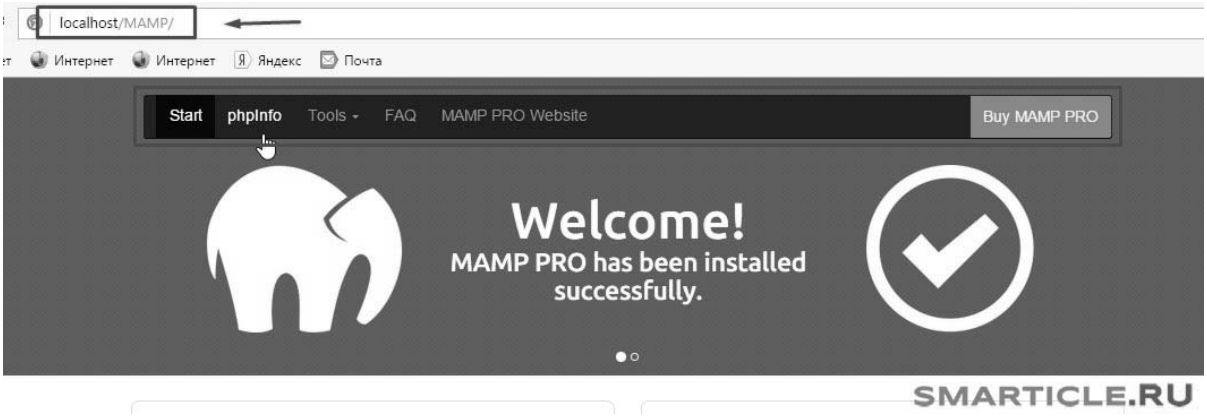

Рисунок 10.3 – Окно браузера

Здесь в адресной строке прописан локальный путь, по которому будет выполняться обращение к файлам сайта – *localhost/MAMP*.

Далее следует навигационное меню, в котором интересен только один раздел Tools (Инструментарий). Именно здесь расположена ссылка для доступа в *phpMyAdmin*.

Порядок выполнения работы.

Выполнить установку и настройку веб-сервера *Apache*, интерпретатора PHP и СУБД MySQL на свой компьютер. Для этого следует ознакомиться с установкой и настройкой CMS MAMP.

Контрольные вопросы

1 Что такое локальный веб-сервер?
2 Для чего используется локальный веб-сервер *Apache*?
3 Что такое CMS?
4 Как установить CMS *WordPress*?
5 Как создать статическую страницу в *WordPress*?
6 Для чего используются пакеты WAMP и MAMP?
7 Как настроить CMS *WordPress* и тему?

11 Лабораторная работа № 11. Изучение строковых функций языка PHP

В языке PHP используются три способа задания строк: с помощью одинарных кавычек, двойных кавычек и с использованием *heredoc*-синтаксиса.

Строки, содержащие заключенные в одинарные кавычки переменные и управляющие последовательности специальных символов, не обрабатываются.

Важнейшим свойством строк в двойных кавычках является обработка содержащихся в них переменных.

Определение строк с использованием *heredoc*-синтаксиса начинается с символа <<<, после которого следует идентификатор. Заканчивается строка этим же идентификатором, который должен начинаться с первой позиции новой строки.

Heredoc-текст ведет себя так же, как и строка в двойных кавычках. Это означает, что в *heredoc* нет необходимости экранировать кавычки, но можно использовать управляющие последовательности. Переменные внутри *heredoc* также обрабатываются.

Для работы со строками в PHP имеется более ста функций (таблица 11.1).

Таблица 11.1 – Некоторые функции обработки строк

Номер функции	Функция
1	2
1	chunk_split – разбивает строку на фрагменты
2	echo – выводит одну или более строк
3	explode – разбивает строку с помощью разделителя
4	implode – объединяет элементы массива в строку
5	lcfirst – преобразует первый символ строки в нижний регистр
6	ltrim – удаляет пробелы или другие символы из начала строки
7	print – выводит строку
8	printf – выводит отформатированную строку
9	rtrim – удаляет пробелы или другие символы из конца строки
10	similar_text – вычисляет степень похожести двух строк
11	sprintf – возвращает отформатированную строку
12	sscanf – разбирает строку в соответствии с заданным форматом
13	stripos – возвращает позицию первого вхождения подстроки без учета регистра
14	strlen – возвращает длину строки
15	str_pad – дополняет строку другой строкой до заданной длины
16	str_replace – заменяет все вхождения строки поиска на строку замены
17	str_shuffle – переставляет символы в строке случайным образом
18	str_split – преобразует строку в массив
19	str_word_count – возвращает информацию о словах, входящих в строку
20	strip_tags – удаляет теги *html* и PHP из строки
21	strpbrk – ищет в строке любой символ из заданного набора
22	strpos – возвращает позицию первого вхождения подстроки
23	strrchr – находит последнее вхождение символа в строке
24	strrev – переворачивает строку задом наперед

Окончание таблицы 11.1

1	2
25	stripos – возвращает позицию первого вхождения подстроки без учета регистра
26	strlen – возвращает длину строки
27	strpbrk – ищет в строке любой символ из заданного набора
28	strpos – возвращает позицию первого вхождения подстроки
29	strrchr – находит последнее вхождение символа в строке
30	strrev – переворачивает строку задом наперед
31	strripos – возвращает позицию последнего вхождения подстроки без учета регистра
32	strrpos – возвращает позицию последнего вхождения подстроки в строке
33	strspn – возвращает длину участка в начале строки, соответствующего маске
34	strstr – находит первое вхождение подстроки
35	strtok – разбивает строку на токены
36	strtolower – преобразует строку в нижний регистр
37	strtoupper – преобразует строку в верхний регистр
38	strtr – преобразует заданные символы или заменяет подстроки
39	substr_count – возвращает число вхождений подстроки
40	substr_replace – заменяет часть строки
41	substr – возвращает подстроку
42	trim – удаляет пробелы или другие символы из начала и конца строки
43	ucfirst – преобразует первый символ строки в верхний регистр
44	ucwords – преобразует в верхний регистр первый символ каждого слова строки
45	vfprintf – записывает отформатированную строку в поток
46	vprintf – выводит отформатированную строку
47	vsprintf – возвращает отформатированную строку
48	wordwrap – переносит строку по указанному количеству символов
49	serialize() – создает строковое представление текущего состояния массива или объекта
50	unserialize() – восстанавливает состояние массива / объекта из строки

Порядок выполнения работы.

1 Работа со строками:

- определить строку с использованием синтаксиса одинарных кавычек;
- определить строку с использованием синтаксиса двойных кавычек;
- определить строку с использованием *heredoc*-синтаксиса;
- создать массив из трех-пяти элементов, вывести его с использованием *echo, print, print_r, serialize* и пояснить полученные результаты.

2 Составить программу на языке PHP с использованием функций, указанных в таблице 11.2, согласно варианту.

Таблица 11.2 – Варианты заданий

Номер варианта	Номер функции	Номер варианта	Номер функции	Номер варианта	Номер функции
1	1, 13, 25, 37	5	5, 17, 29, 41	9	9, 21, 33, 45
2	2, 14, 26, 38	6	6, 18, 30, 42	10	10, 22, 34, 46
3	3, 15, 27, 39	7	7, 19, 31, 43	11	11, 23, 35, 47
4	4, 16, 28, 40	8	8, 20, 32,44	12	12, 24, 36, 48

Контрольные вопросы

1 Каковы особенности строк, записанных в одинарных кавычках?
2 В чем особенности строк, записанных в двойных кавычках?
3 Что понимается под *heredoc*-синтаксисом?
4 Как объединить элементы массива в строку?
5 Как удалить пробелы или другие символы из конца строки?
6 Как преобразовать строку в массив?
7 Какая функция переворачивает строку задом наперед?

12 Лабораторная работа № 12. Изучение операторов цикла языка PHP

В языке PHP существует несколько конструкций, позволяющих выполнять повторяющиеся действия в зависимости от условия. Это циклы *while*, *do ...while, foreach u for*.

while – это простой цикл. Он имеет две формы записи:

while (выражение) { блок_выполнения }

либо

while (выражение): блок_выполнения *endwhile*;

Циклы *do...while* похожи на циклы *while*, но в них истинность выражения проверяется в конце цикла. Форма записи:

do {блок_выполнения} *while* (выражение);

Цикл *for* со счетчиком используется для выполнения тела цикла определенного числа раз.

Синтаксис цикла *for*

for (*инициализирующие_команды; условие; команды_после_итерации*) { *тело_цикла*; }

Цикл *for* начинает свою работу с выполнения инициализирующих команд, которые выполняются только один раз. Затем проверяется условие цикла, и если оно истинно (*true*), то выполняется тело цикла. После того как будет выполнен последний оператор тела цикла, выполняются команды после итерации. Затем снова проверяется условие цикла. Если оно истинно (*true*), выполняется тело цикла и команды после итерации и т. д. Например:

```
<?php
for ($x=0; $x<10; $x++) echo $x;
?>                  // выводит: 0123456789
```

Если необходимо указать несколько команд, то их можно разделить запятыми, например:

```
<?php
for ($x=0, $y=0; $x<10; $x++, $y++) echo $x;
```

```
?>                    // Выводит 0123456789
```

Пример использования нескольких команд в цикле *for*

```
<?php
for($i=0,$j=0,$k="Точки"; $i<10; $j++,$i+=$j) { $k=$k."."; echo $k; }
                    // Выводит Точки.Точки..Точки...Точки....
?>
```

Цикл *for* имеет альтернативный синтаксис

```
for(инициализирующие_команды; условие; команды_после_итерации);
операторы;
endfor;
```

Цикл *foreach* перебора массивов имеет синтаксис

```
foreach (массив as $ключ=>$значение)
команды;
```

Порядок выполнения работы.

Написать и отладить скрипт, выполняющий действия, указанные в таблице 12.1, согласно варианту.

Таблица 12.1 – Варианты заданий

Номер варианта	С помощью цикла
1	Найти сумму корней чисел от 1 до 15. Результат округлить до двух знаков в дробной части
2	Найти сумму тех чисел от 1 до 100, которые делятся на 7
3	Создать строку из шести символов, состоящую из случайных чисел от 1 до 9
4	Дан массив с числами. Найти сумму квадратов его элементов
5	Дан массив с числами. Найти корень из суммы квадратов элементов этого массива, а результат округлить в меньшую сторону до целых
6	Дан массив с числами. Найти сумму тех его чисел, которые больше 0 и меньше 10
7	Заполнить двумерный массив, содержащий 10 подмассивов, случайными числами от 1 до 10. В каждом подмассиве должно быть по 10 элементов
8	Преобразовать строку 'var_text_hello' в 'varTextHello'. Скрипт должен работать с любыми строками такого типа
9	Дан массив с произвольными числами. Сделать так, чтобы элемент в массиве повторился количество раз, соответствующее его значению. Например, [1, 3, 2, 4] должен превратиться в [1, 3, 3, 3, 2, 2, 4, 4, 4, 4]
10	Дана строка. Удалить из этой строки четные символы
11	Дана строка. Поменять ее первый символ на второй и наоборот, третий на четвертый и наоборот, пятый на шестой и наоборот и т. д., т. е. из строки '12345678' необходимо сформировать строку '21436587'
12	Написать скрипт, который проверяет, являются ли заданные числа простыми, то есть делящимися только на единицу и сами на себя

Контрольные вопросы

1 Какие операторы цикла Вы знаете?
2 Какие формы записи конструкции *while* Вы знаете?
3 Прокомментируйте работу цикла *do..while*.
4 Для чего используется конструкция *for*?
5 Какие формы записи конструкции *for* Вы знаете?
6 Для чего используется оператор *break* в цикле *for*?
7 Прокомментируйте работу конструкции *foreach*.

13 Лабораторная работа № 13. Изучение приемов работы с массивами на языке PHP

В языке PHP в одном массиве допускается хранение переменных различных типов, а также массивов и объектов. Для обращения к элементу массива используется его индекс (ключ).

PHP поддерживает работу с индексными и ассоциативными массивами, индексами которых являются строки.

Для обращения к элементам индексных массивов используются числовые индексы, а ассоциативных – строковые.

Для создания массивов можно использовать конструкцию *array*() или способ приведения скалярной переменной типа *int, float, string* или *boolean* к типу *array*, а также специализированные функции:

array([…]) – создает массив из значений, переданных конструкции в качестве параметров *array_fill($start_index*, *$num*, *$value*), которая возвращает массив, содержащий *$num* элементов, имеющих значение *$value*. Нумерация индексов при этом начинается со значения *$start_index*;

range($low, $high [, $step]) – создает массив со значениями из интервала от *$iow* до *$high* и шагом *$step*;

explode($delimiter, *$str [, $limit*) – возвращает массив из строк, каждая из которых соответствует фрагменту исходной строки *$str*, находящемуся между разделителем, определяемым аргументом *$delimiter*. Необязательный параметр *$limit* определяет максимальное количество элементов в массиве, при этом последний элемент будет содержать остаток строки *$str*.

В качестве элементов массива могут выступать другие массивы, в этом случае говорят о многомерных массивах. Массивы можно создавать, обращаясь к элементам или используя вложенные конструкции *array()*. Для вывода массива используется функция *print_r().*

Работу с ассоциативными массивами удобно выполнять с использованием специализированного оператора цикла *foreach*.

При манипуляции с массивами и их элементами часто возникает необходимость определения количества элементов в массиве. Для решения этой задачи используются следующие функции:

count($array [,$mode]) – возвращает количество элементов массива *$array*. Если *$mode* принимает значение *count_recursive*, функция рекурсивно обходит многомерный массив, в противном случае подсчитывается количество элементов только на текущем уровне;

sizeof() – синоним для функции *count()*;

array_count_values($input) – подсчитывает количество уникальных значений среди элементов массива и возвращает ассоциативный массив, ключами которого являются значения массива, а значениями – количество их вхождений в массив *$input*.

Порядок выполнения работы.

Написать и отладить скрипт, выполняющий действия, указанные в таблице 13.1, согласно варианту.

Таблица 13.1 – Варианты заданий

Номер варианта	Задание
1	В массиве из n строк проверить, начинается ли каждая строка символом "*", а строки без "*" перенести в другой массив
2	В массиве из n строк проверить, содержит ли k-я строка символ @. Если не содержит, то вставить этот символ в конец строки
3	В массиве строк удалить все *html*-теги, заключенные в скобки < >
4	В массив случайным образом поместить строки, содержащие «цитата дня». Для выбора строки из массива случайным образом можно использовать функции *Shuffle(array arr)* или *arrayrand(array arr, int num)*
5	Создать многомерный массив: Факультет, Курс, Группа, Студенты. Вывести список студентов в алфавитном порядке
6	Создать многомерный массив: Факультет, Кафедра, Преподаватель, Ученое звание. Вывести список преподавателей в алфавитном порядке
7	Создать двухмерный массив, в первой строке которого записаны номера и названия месяцев года в произвольном порядке. Во второй строке рассортировать месяцы года в алфавитном порядке, а в третьей – в порядке возрастания номера месяца
8	Заполнить элементы квадратной матрицы возрастающими числами начиная с единицы по спирали, начиная с элемента [1, 1] по часовой стрелке
9	Заполнить элементы квадратной матрицы возрастающими числами начиная с единицы по спирали, начиная с элемента [n, n] против часовой стрелки
10	Перемножить две числовые матрицы, размеры и значения матриц выбрать самостоятельно
11	Найти максимальную сумму диагональных элементов квадратной матрицы, размер и значения матрицы выбрать самостоятельно
12	Найти суммы элементов двух квадратных матриц и выбрать матрицу с большей суммой, размеры и значения матриц выбрать самостоятельно

Контрольные вопросы

1 Допускается ли хранение в одном массиве значений разных типов?
2 Что такое ассоциированный массив?

3 Какие конструкции используются для создания массивов?
4 Что такое индексный массив?
5 Как создать двухмерный массив?
6 Для чего используется функция *Shuffle(array arr)*?
7 Прокомментируйте назначение функции *arrayrand(array arr, int num)*.

14 Лабораторная работа № 14. Изучение технологии работы с функциями PHP

В PHP существуют две основные формы функций: встроенные и пользовательские.

Полный список встроенных PHP-функций можно просмотреть в окне редактора кода, нажав кнопку «*Поиск*» в правой колонке при пустой строке поиска «PHP-поиск». Для просмотра подробного описания с примером конкретной PHP-функции необходимо указать ее имя в строке PHP-поиск.

Пользовательская функция создается с помощью оператора *function*, после которого через пробел указывается имя функции и круглые скобки, которые могут быть пустыми либо содержать параметры, переменные PHP, принимаемые функцией.

После объявления функции в фигурных скобках записывается код, выполняемый функцией. Синтаксис пользовательской функции имеет вид:

```
function имя_функции (параметры)
{
  //тело функции
}
```

Вызывается функция по ее имени, после которого обязательны круглые скобки, даже если они пустые.

Для возврата значения, являющегося результатом работы функции, используется оператор *return*, который прекращает выполнение функции, возвращает ее значение и может быть расположен в любом ее месте.

В PHP допускается использование динамических функций. Это означает, что если некоторой переменной присвоено имя функции, то с этой переменной можно обращаться точно так же, как с самой функцией.

В функциях допускается использование глобальных переменных, созданных с помощью инструкции *global* вне функции.

Чтобы переменная сохраняла свое значение между вызовами функции, нужно объявить ее статической с помощью инструкции *static*.

Порядок выполнения работы.

Написать и отладить скрипт, выполняющий действия, указанные в таблице 14.1, согласно варианту.

Таблица 14.1 – Варианты заданий

Номер варианта	Задания
1	2
1	Создать пользовательскую функцию, которая принимает два аргумента и возвращает их произведение. Вызвать функцию, передав ей в качестве аргументов два числа, и результат вывести на экран. Создать пользовательскую функцию, формирующую массив делителей (чисел, на которое оно делится без остатка) заданного числа
2	Создать три переменные, присвоить им числовые значения и вывести на экран их произведение. Создать пользовательскую функцию, принимающую два аргумента по ссылке и один по значению, которая должна присваивать переменным другие числовые значения. Вызвать функцию и вывести на экран произведение всех переменных
3	Создать две переменные, присвоить им числовые значения и создать пользовательскую функцию, принимающую два аргумента со значениями по умолчанию и выводящую произведение своих аргументов. Вызвать функцию, передав ей в качестве аргументов сначала значения двух переменных, затем значение одной из переменных и, наконец, вообще без аргументов
4	Создать пользовательскую функцию, принимающую аргументы в массив переменной длины и выводящую их на экран. Для доступа к элементам массива использовать цикл *foreach*. Вызвать функцию, передав ей в качестве значения две строки и число
5	Создать пользовательскую функцию, которая вычисляет корень из заданного четырехзначного числа и округляет его в большую и меньшую стороны. В массив *$arr* первым элементом записать заданное число, вторым – корень из этого числа, третьим – округление в меньшую сторону, четвертым – округление в большую сторону
6	Заполнить массив 30 случайными числами от 1 до 10. Найти квадратный корень из каждого числа. Округлить результаты в большую и меньшую сторону и записать результаты округления в ассоциативный массив с ключами 'floor' и 'ceil'.
7	Дано целое двухзначное число. Создать функцию, определяющую делители этого числа, т. е. числа, на которое оно делится без остатка. Сформировать массив делителей заданного числа
8	Задать режим строгой типизации, использовать инструкцию *declare(strict_types*=1), после этого создать пользовательскую функцию, которая будет принимать два целочисленных аргумента и выводить на экран их сумму. Вызвать функцию, передав ей в качестве аргументов сначала два целых числа, а затем одно из них в виде строки
9	Задать режим строгой типизации, используя инструкцию *declare(strict_types*=1), затем создать пользовательскую функцию *my_func*(), которая будет принимать два целочисленных аргумента и возвращать их произведение. Создать переменную *$count_apples* и присвоить ей строку с именем функции. Обратиться к функции через переменную и вывести на экран общую массу яблок, зная, что имеется 25 корзин по 7 кг яблок в каждой
10	Создать переменную и присвоить ей целое число. Создать еще одну переменную и присвоить ей анонимную функцию, наследующую эту переменную и выводящую на экран ее инкрементированное значение. Выполнить вызов функции, затем изменить значение внешней переменной и снова вызвать функцию. Изменить скрипт, задав наследование переменной по ссылке

Окончание таблицы 14.1

1	2
1	Создать функцию, вычисляющую квадратный корень из заданного числа. Результат округлить в большую и меньшую сторону, результаты округления записать в ассоциативный массив с ключами "floor" и "ceil". Создать функцию, определяющую, сколько первых элементов массива [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] нужно сложить, чтобы сумма получилась больше 10
2	Создать пользовательскую функцию, вычисляющую корень квадратный из заданного четырехзначного числа. Результат округлить в большую и меньшую стороны. В массив *$arr* записать первым элементом корень из числа, вторым – округление в меньшую сторону, третьим – округление в большую сторону. Для решения задачи можно использовать функции: *sqrt* – корень из числа, *floor* – округление в меньшую сторону, *ceil* – округление в большую сторону

Контрольные вопросы

1 Какие две основные формы PHP-функций Вы знаете?
2 Как посмотреть полный список встроенных PHP-функций?
3 Как просмотреть подробное описание конкретной PHP-функции?
4 Как создаются пользовательские PHP-функции?
5 Что записывается в фигурных скобках пользовательской PHP-функции?
6 Прокомментируйте общий синтаксис PHP-функции.
7 Как вызвать PHP-функцию?

15 Лабораторная работа № 15. Изучение технологии работы с файлами PHP

В PHP-документ с помощью инструкции *include()* можно включать файлы кода. Аргументом этой инструкции является путь к файлу. Чтобы содержимое этого файла обрабатывалось как PHP-программа, его необходимо обрамлять открывающим и закрывающим тегами PHP.

PHP содержит множество функций управления файлами, наиболее употребительными из которых являются:

touch() – создает пустой файл с заданным именем, например: *touch("ex1.txt")*. Если такой файл уже существует, то функция изменит дату модификации;

copy() – копирует файл. Для копирования файлов в PHP используется функция *copy($source, $result)*, в которой используются два параметра – источник *$source* и имя файла-копии *$result*. При этом следует указывать полные адреса к файлам;

unlink() – удаляет заданный файл. Например:

```
<?php
if (unlink('filename.txt'))
```

```
    { echo "Файл удален"; }
else
    { echo "Ошибка при удалении файла"; }
?>
```

fopen() – открывает локальный или удаленный файл и возвращает указатель на него. Указатель используется во всех операциях с содержимым файла. Аргументами функции *fopen()* являются имя файла и режим открытия;

fclose() – закрывает файл. Аргумент: указатель файла, полученный ранее от функции *fopen()*;

feof() – проверяет конец файла. Аргумент: указатель файла;

fgetc() – читает очередной символ из файла. Аргумент: указатель файла;

fgets() – читает очередную строку файла. Аргументы: указатель файла и длина считываемой строки. Операция прекращается после считывания заданного количества символов или при обнаружении конца строки или файла;

fread() – общая функция чтения из файла. Аргументы: указатель файла и количество считываемых символов;

fseek() – отступ от начала файла. Аргументы: указатель файла и смещение;

fputs() – записывает строку в файл. Аргументы: указатель файла и строка;

fwrite() – полный аналог функции *fputs( )*;

flock() – блокирует файл, т. е. не позволяет другим пользователям читать этот файл или писать в него, пока тот, кто наложил блокировку, не закончит работу с данным файлом. Аргументы: указатель файла и номер режима блокировки.

Порядок выполнения работы.

Написать и отладить скрипт, выполняющий действия, указанные в таблице 15.1, согласно варианту.

Таблица 15.1 – Варианты заданий

Номер варианта	Задание
1	Дан массив со строками. Создать папку test, а в ней – папки, названиями которых служат элементы этого массива
2	Найти все файлы из папки test и вставить в начало каждого файла полный путь к нему. Текст файла должен остаться в нем и начинаться с новой строки после пути
3	Вывести на экран имена всех папок из папки test и их подпапок. Уровень вложенности папок может быть любым
4	Вывести на экран содержимое всех файлов из папки test и ее подпапок. Уровень вложенности папок может быть любым
5	Найти все файлы из папки test и ее подпапок любого уровня вложенности и вставить в начало каждого файла полный путь к нему (текст файла должен остаться в нем и начинаться с новой строки после пути)
6	Удалить из папки test все файлы размером более 1 Мбайт
7	Имеется папка с файлами, определить и вывести на экран размер этой папки
8	Имеется папка с подпапками, определить размеры всех подпапок папки и вывести их на экран

Окончание таблицы 15.1

1	2
9	Вывести на экран название всех файлов с расширением *txt* из папки test
10	В папке test есть файлы и подпапки. Вывести на экран содержимое всех файлов, которые лежат непосредственно в папке test
11	Вывести на экран название всех файлов, но не подпапок из папки test
12	Вывести на экран название всех файлов и подпапок из папки test

Контрольные вопросы

1 С помощью какой инструкции можно включать файлы кода в PHP-документ?

2 С помощью какой PHP-функции можно создать пустой файл?

3 Какая PHP-функция используется для открытия файла?

4 Какие параметры используются в PHP-функции *copy()*?

5 Какая PHP-функция используется для чтения очередной строки файла?

6 Какая PHP-функция используется для записи строки в файл?

7 Для чего используется PHP-функция *fputs()*?

16 Лабораторная работа № 16. Ознакомление с командным интерпретатором и основными консольными командами ОС Linux

Изучение ОС *Linux* целесообразно начинать с освоения средств ее командной оболочки, т. к. многие действия *Linux* гораздо быстрее и эффективнее выполнить, используя только командную строку.

Основными элементами ОС *Linux* являются файлы, например, текстовые документы – это файлы, изображения, аудиоданные в формате mp3 и видеофрагменты, а также каталоги, содержащие информацию о других файлах – это все файлы.

Дисковые устройства и сетевые соединения – это большие файлы. Даже исполняемый процесс также является файлом. То есть в ОС *Linux* файл представляет собой поток битов или байтов.

В отличие от Windows и MacOS в операционной системе *Linux* имена файлов чувствительны к регистру символов.

С точки зрения файловой операционной системы *Linux* – это различные имена файлов. В названиях файлов не рекомендуется использовать следующие специальные символы: /; \, -, [,], {, }, *, ?, ', ".

Для работы с файлами в ОС *Linux* можно использовать символы групповых операций, которые задаются посредством звездочки (*), знака вопроса (?) и квадратных скобок ([ ]). Например символ (*) в имени файла означает нуль и более символов, знак (?) – один произвольный символ, а групповая операция с применение символов квадратных скобок ([ ]) позволяет задавать один символ из набора или диапазона символов.

Основные команды, используемые в ОС *Linux*:

$ pwd – определить текущий каталог;

$ cd [*имя каталога*] – осуществить переход в заданный каталог;

$ ls [*имя каталога*] – просмотреть список файлов и подкаталогов;

$ mkdir [*имя каталога*] – создать каталог с заданным именем;

$ cp <имя файла 1> *<имя файла 2>* – скопировать файл «имя файла 1» в файл «имя файла 2», например: cp first.txt copy1.txt;

$ mv <имя файла 1> <имя файла 2> – переименовать файл «имя файла 1» в файл «имя файла 2», например: mv first.txt orig.txt;

$ ln «*имя файла*» «*имя ссылки*» – создать жёсткую ссылку «имя ссылки» на файл «имя файла», например: ln orig.txt copy2.txt;

$ ln -s «*имя файла*» «*имя ссылки*» – создать символическую ссылку «имя ссылки» на файл «имя файла», например: ln -s orig.txt copy2.txt;

$ touch <имя файла> – создание файла;

$ man <название команды> – получение справочной документации о выбранной команде.

Порядок выполнения работы.

Перед выполнением каждого задания необходимо ознакомиться с ее возможностями с помощью команды *man*. После выполнения пунктов 4, 15 и 16 необходимо сделать копию экрана для отчета по лабораторной работе.

1 Открыть терминал.

2 Определить текущий каталог, в котором Вы находитесь, командой pwd.

3 Перейти в корневой каталог командой cd.

4 Просмотреть содержимое корневого каталога командой ls.

5 Вернуться в домашний каталог, используя команду cd без параметров.

6 Создать каталог «test», используя команду mkdir.

7 Перейти в каталог «test», используя команду cd.

8 Просмотреть содержимое каталога «test», используя команду ls.

9 Создать каталог «test2», используя команду mkdir.

10 Создать файл «text» в каталоге «test2», используя команду touch.

11 Переименовать файл «text» в «textSIT», используя команду mv.

12 Скопировать файл «textSIT» в каталог «test2» под именем «copy.txt», используя команду cp.

13 Создать жесткую ссылку «*link*» на файл «copy.txt», используя команду ln.

14 Создать символическую ссылку «*simlink*» на файл «copy.txt», используя команду ln.

15 Просмотреть результаты в текущем каталоге при помощи команды ls с аргументами la.

16 Удалить созданные файлы и ссылки в лабораторной работе, используя команду *rm*.

Контрольные вопросы

1 Что такое «жесткая ссылка»?
2 Чем отличается вывод команд ls -F и ls -la?
3 Что такое «символическая ссылка»?
4 Как осуществить просмотр подкаталогов и их содержимого ?
5 С помощью какой команды можно переместить файл в другой каталог?
6 Как осуществить просмотр скрытых файлов в домашнем каталоге?
7 Как создать новый каталог и необходимые подкаталоги рекурсивно?

17 Лабораторная Работа № 17. Изучение ОС Android и ее установка

Android – это основанная на ядре *Linux* бесплатная операционная система для смартфонов, планшетов, электронных книг, цифровых проигрывателей, наручных часов, игровых приставок, ноутбуков, нетбуков, телевизоров.

Платформа *Android* объединяет три компонента: операционную систему, промежуточное программное обеспечение и встроенные мобильные приложения. Она содержит библиотеку элементов пользовательского интерфейса, поддерживает 2D и 3D-графику, а также доступ к файловой системе и встроенной базе данных *SQLite*.

С точки зрения архитектуры система *Android* представляет собой полный программный стек, содержащий следующие уровни:

– базовый (*Linux Kernel*) уровень абстракции между аппаратным уровнем и программным стеком;

– набор библиотек и среда исполнения (*Libraries & Android Runtime*), обеспечивающая базовый функционал для приложений, содержит виртуальную машину *Dalvik* и базовые библиотеки *Java*, необходимые для запуска *Android* приложений;

– каркас приложений *Application Framework*, предоставляющий разработчикам доступ к API;

– набор предустановленных базовых приложений *Applications*.

Изображение архитектуры ОС *Android* приведено на рисунке 17.1.

В основе компонентной иерархии *Android* лежит несколько урезанное ядро ОС *Linux 2.6*, которое служит промежуточным уровнем между аппаратным и программным обеспечением. Это ядро обеспечивает функционирование системы, предоставляет такие его системные службы, как управление памятью, энергосистемой и процессами,, обеспечение безопасности, работа с сетью и драйверами.

В уровне выше располагается среда исполнения и набор библиотек, которые реализуют алгоритмы для вышележащих уровней, выполняют поддержку файловых форматов, кодирование и декодирование информации, отрисовку графики и т. д. Основными библиотеками этого набора являются:

– *Surface Manager*, представляющая композитный менеджер окон, позволяющий системе создавать интересные бесшовные эффекты, прозрачность окон и плавные переходы;

– *Media Framework*, использующаяся для записи и воспроизведения аудио- и видеоконтента, а также для вывода статических изображений форматов MPEG4, H.264, MP3, AAC, AMR, JPG и PNG;

– *SQLite* – реляционная СУБД, используемая в качестве основного движка для работы с базами данных;

– *3D-библиотеки* – высокооптимизированные отрисовки 3D-графики, реализованные на основе API OpenGL|ES;

– *LibWebCore* – библиотека браузерного движка WebKit, используемая в браузерах Google Chrome и Apple Safari;

– *SGL* (*SkiaGraphicsEngine*) – открытый движок для работы с 2D-графикой;

– *SSL* – поддержки одноименного криптографического протокола;

– *Libc* – стандартная библиотека языка C, настроенная для работы на базе *Linux*.

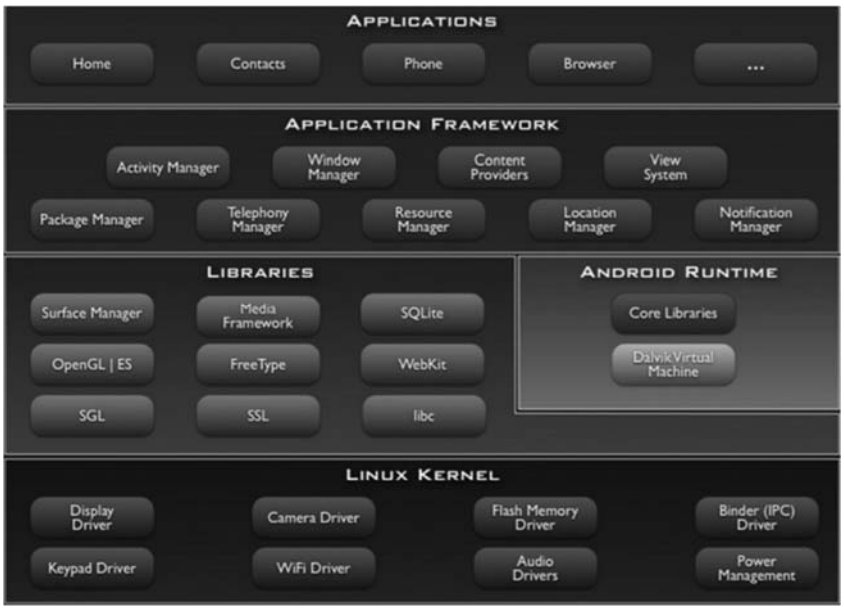

Рисунок 17.1 – Архитектура ОС ***Android***

Эти библиотеки реализованы на языке C/C++ и поставляются в предустановленном виде.

Среда исполнения *Android* включает в себя библиотеки ядра, обеспечивающие большую часть низкоуровневой функциональности, доступной библиотекам ядра языка *Java*, и виртуальную машину *Dalvik*, позволяющую запускать приложения.

Каждое приложение запускается в своем экземпляре виртуальной машины, тем самым обеспечивается изоляция работающих приложений от ОС и друг от друга.

Для исполнения на виртуальной машине *DalvikJava*-классы компилируются в исполняемые файлы с расширением *.dex* с помощью инструмента *dx*, входящего в состав *Android SDK*.

Расширение *.dex* (*Dalvik EXecutable*) – формат исполняемых файлов для виртуальной машины *Dalvik*, оптимизированный для использования минимального объема памяти. При использовании *IDE Eclipse* и плагина *ADT* (*Android Development Tools*) компиляция классов *Java* в формат *.dex* происходит автоматически.

Архитектура *Android Runtime* такова, что работа программ осуществляется строго в рамках окружения виртуальной машины, что позволяет защитить ядро ОС от возможного вреда со стороны других ее составляющих.

На еще более высоком уровне располагается каркас приложений *Application Framework*, архитектура которого позволяет любому приложению использовать уже реализованные возможности других приложений, к которым разрешен доступ. В состав каркаса входят следующие компоненты:

– расширяемый набор представлений *Views*, который может использоваться для создания визуальных элементов, таких как списки, текстовые поля, таблицы, кнопки и даже встроенный веб-браузер;

– контент-провайдеры (*Content Providers*), управляющие данными, которые одни приложения открывают для других, чтобы те могли использовать их для своей работы;

– менеджер ресурсов (*Resource Manager*), обеспечивающий доступ к ресурсам без функциональности, т. е. не несущим кода, например, к строковым данным, графике и другим;

– менеджер оповещений (*Notification Manager*), позволяющий приложениям отображать собственные уведомления для пользователя в строке состояния;

– менеджер действий (*Activity Manager*), управляющий жизненными циклами приложений, сохраняющий историю работы с действиями, предоставляющий систему навигации по действиям;

– менеджер местоположения (*Location Manager*), позволяющий приложениям периодически получать обновленные данные о текущем географическом положении устройства.

И, наконец, самый высокий, самый близкий к пользователю уровень приложений. Именно на этом уровне пользователь взаимодействует со своим устройством, управляемым ОС *Android*. Здесь представлен набор базовых приложений, который предустановлен на ОС *Android*. Например, браузер, почтовый клиент, программа для отправки SMS, карты, календарь, менеджер контактов и др. Список интегрированных приложений может меняться в зависимости от модели устройства и версии *Android*. К этому уровню также относятся все пользовательские приложения. Разработчик обычно взаимодействует с двумя верхними уровнями архитектуры *Android* для создания новых приложений. Библиотеки, система исполнения и ядро *Linux* скрыты за каркасом приложений. Повторное использование компонентов других

приложений приводит к идее задач в *Android*. Приложение может использовать компоненты другого *Android*-приложения для решения задачи, например, если разрабатываемое приложение предполагает использование фотографий, оно может вызвать приложение, управляющее фотографиями и зарегистрированное в системе *Android*, выбрать с его помощью фотографию и работать с ней.

Пополнение коллекции приложений мобильного устройства можно выполнять с помощью приложений сервиса *Google Play*. Разработчики могут выкладывать свои приложения в этот сервис, а *Google Play* отслеживает появление обновлений, сообщает об этом пользователям и предлагает его установить. Также *Google Play* предоставляет разработчикам доступ к услугам и библиотекам, например, доступ к использованию и отображению *Google Maps* – комплекса приложений, созданных на базе бесплатного сервиса картографии и используемого для поиска информации на карте с отметками достопримечательностей, организаций и т. д.

Для установки приложения на устройствах с ОС *Android* создается файл с расширением **.apk* (*Android package*), который содержит исполняемые файлы, а также вспомогательные компоненты, например, файлы с данными и файлы ресурсов. После установки на устройство каждое приложение сохраняется в собственном изолированном экземпляре виртуальной машины *Dalvik*.

Главная особенность *Dalvik* в том, что ее приложение полностью компилируется при открытии или запуске пользователем соответствующих инструкций.

Google представила новую виртуальную машину *ART* (*Android RunTime*) в *Android* 4.4, которая компилирует все инструкции приложения в процессе установки, т. е. до его запуска.

Разработку новых *Android*-приложений, позволяющих расширить функциональные возможности смартфона, можно выполнить с использованием инструментов и интерфейсов прикладного программирования API (*Application Programming Interface*) с использованием языка *Java*. Однако выполнять отладку приложений на смартфоне неудобно, поэтому разработаны специальные программные средства для таких работ на ПК в среде ОС *Linux* и даже эмуляторы *Android* для Windows, которые можно использовать для игр или для тестирования своих программ.

ОС *Android* можно установить на компьютеры и ноутбуки, работающие под управлением Windows и OS X, или в виде самостоятельной либо второй операционной системы без использования виртуальных машин.

При установке *Android* в качестве второй операционной системы ПК обеспечивается значительное увеличение быстродействия мобильной ОС, т. к. оперативная память и процессор не загружаются дополнительными приложениями или эмуляторами.

Для установки ОС *Android* на ПК необходимо наличие образа *Android*, утилиты *Rufus* для создания загрузочной флешки, флешки с объемом памяти не меньше 16 ГБ, компьютера или ноутбука.

После создания загрузочной флешки и скачивания на нее ISO-образа *Android* разрядности ПК (x64 или x32) необходимо перезагрузить ПК.

Запуск установки с флешки начинается с установки флеш-накопителя с образом *Android* и загрузки с нее компьютера.

Далее, если подготовка установки была выполнена верно, то загрузится меню установки *Android* (рисунок 17.2).

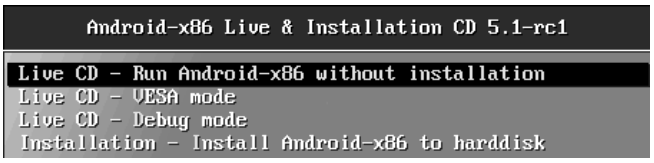

Рисунок 17.2 – Меню установки *Android*

Выберем первый пункт (запуск без установки), дождемся завершения процесса загрузки, после чего появится первоначальное окно настроек, после которых отобразится рабочий стол, готовый к использованию ОС *Android* (рисунок 17.3).

Рисунок 17.3 – Рабочий стол ОС *Android*

Установка ОС *Android* на компьютер занимает значительно больше времени и может быть выполнена двумя способами: в качестве основной системы либо в качестве второй дополнительной ОС.

Для установки потребуются USB-накопитель объемом не менее 1 ГБ и отдельный дисковый раздел размером не менее 8 ГБ. Дисковый раздел для установки *Android* можно создать стандартными средствами операционной системы, используя контекстное меню кнопки «Пуск» – «Управление дисками» для Windows.

Порядок выполнения работы.

1 Изучить основные положения по назначению и использованию ОС *Android*, используя ресурсы интернета или литературные источники.

2 Оформить отчет с найденными определениями и положениями со ссылками на источники.

Контрольные вопросы

1 Какие ОС используются в современных портативных устройствах?
2 Какие инструменты используются для разработки *Android*-приложений?
3 Для чего смартфон на *Android* содержит ядро *Linux*?
4 Возможна ли установка ОС *Android* на ПК, работающие под *Windows*?
5 Какие преимущества установки ОС *Android* в качестве второй операционной системы ПК?
6 Осуществляется ли загрузка ОС *Android* с флеш-накопителя?
7 Как создать дисковый раздел для установки *Android* на ПК?

Список литературы

1 **Никольский, А. П.** JavaScript на примерах / А. П. Никольский. – Москва: Наука и техника, 2017. – 274 с.
2 **Вуд, К.** Расширение библиотеки jQuery / К. Вуд. – Москва: ДМК Пресс, 2018. – 184 с.
3 **Макфарланд, Д.** JavaScript и jQuery. Исчерпывающее руководство (+ DVD-ROM) / Д. Макфарланд. – Москва: Эксмо, 2017. – 688 с.
4 **Роббинс, Д.** HTML5, CSS3 и JavaScript. Исчерпывающее руководство (+ DVD-ROM) / Дженнифер Роббинс. – Москва: Эксмо, 2018. – 528 с.
5 **Хэррон, Д.** Node.js Разработка серверных веб-приложений на JavaScript / Дэвид Хэррон. – Москва: ДМК Пресс, 2016. – 862 с.
6 **Валади, Дж.** 100 % самоучитель Linux / Дж. Валади. – Москва: Техно-лоджи-3000, 2018. – 336 с.
7 **Колисниченко, Д. Н.** Linux. Полное руководство / Д. Н. Колисниченко, П. В. Аллен. – Москва; Санкт-Петербург: Наука и Техника, 2017. – 784 с.
8 **Роббинс, А.** Linux: программирование в примерах / А. Роббинс. – Москва: КУДИЦ-Образ, 2018. – 611 с.
9 **Собель, М.** Linux**.** Администрирование и системное программирование / М. Собель. – Москва: Питер, 2016. – 820 с.
10 **Леонов, В.** Android. Планшеты и смартфоны. Простой и понятный самоучитель / В. Леонов. – Москва: Эксмо, 2017. – 212 с.
11 **Голощапов, А.** Google Android. Создание приложений для смартфонов и планшетных ПК / А. Голощапов. – Санкт-Петербург: БХВ-Петербург, 2013. – 832 с.
12 **Кузнецов, М.** Самоучитель PHP 7 / М. Кузнецов, И. Симдянов. – Санкт-Петербург: БХВ-Петербург, 2018. – 450 с.