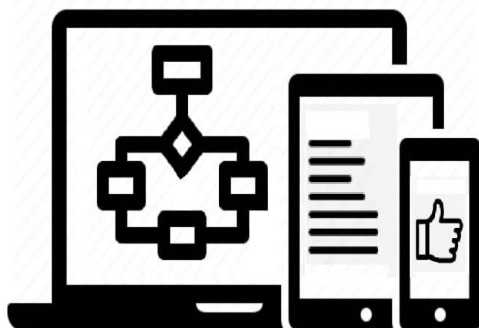


МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение информационных технологий»

# ПРОГРАММИРОВАНИЕ СЕТЕВЫХ ПРИЛОЖЕНИЙ

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



Могилев 2022

УДК 004.43  
ББК 32.973.26-018.1  
П783

Рекомендовано к изданию  
учебно-методическим отделом  
Белорусско-Российского университета

Одобрено кафедрой «Программное обеспечение информационных технологий» «31» января 2022 г., протокол № 7

Составители: канд. техн. наук, доц. Э. И. Ясюкович;  
канд. техн. наук, доц. В. М. Ковальчук

Рецензент канд. техн. наук, доц. М. Н. Миронова

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	В. В. Кутузов
Корректор	А. А. Подошевка
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 21 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, г. Могилев.

© Белорусско-Российский  
университет, 2022

## Содержание

Введение.....	4
1 Лабораторная работа № 1. Разработка веб-страниц на языке html.....	5
2 Лабораторная работа № 2. Использование CSS при разработке веб-страниц .....	8
3 Лабораторная работа № 3. Создание сайта, содержащего сценарии на языке JavaScript.....	12
4 Лабораторная работа № 4. Изучение операторов ветвлений и циклов JavaScript .....	14
5 Лабораторная работа № 5. Изучение функций и методов JavaScript .....	17
6 Лабораторная работа № 6. Изучение приемов работы с массивами JavaScript .....	19
7 Лабораторная работа № 7. Изучение приемов работы с элементами управления JavaScript.....	22
8 Лабораторная работа № 8. Изучение основных методов JQuery .....	24
9 Лабораторная работа № 9. Изучение основных событий JQuery .....	26
10 Лабораторная работа № 10. Установка локального сервера MAMP.....	28
11 Лабораторная работа № 11. Изучение строковых функций языка PHP .....	31
12 Лабораторная работа № 12. Изучение операторов цикла языка PHP .....	33
13 Лабораторная работа № 13. Изучение приемов работы с массивами на языке PHP.....	35
14 Лабораторная работа № 14. Изучение технологии работы с функциями PHP .....	37
15 Лабораторная работа № 15. Изучение технологии работы с файлами PHP .....	40
16 Лабораторная работа № 16. Ознакомление с командным интерпретатором и основными консольными командами ОС Linux .....	41
17 Лабораторная работа № 17. Изучение ОС Android и ее установка .....	43
Список литературы .....	48

## Введение

Цель изучения дисциплины «Программирование сетевых приложений» заключается в освоении теоретических сведений о видах и назначении инструментальных средств разработки веб-приложений, а также получение практических навыков их разработки.

Методические рекомендации содержат указания по организации и выполнению всех видов лабораторных работ, предусмотренных для изучения указанной дисциплины студентами очной и заочной формы обучения по специальности 1-28 01 02 «Электронный маркетинг».

В методических рекомендациях приведены состав, объем, сроки, виды контроля и средства оценки результатов выполнения заданий самостоятельной работы, а также вопросы для самопроверки и примеры заданий.

В современных технологиях веб-программирования широко используются язык разметки гипертекста *html* (*Hyper Text Markup Language*), каскадные таблицы стилей *CSS* (*Cascading Style Sheets*), 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 др.

Для упрощения процесса разработки *JavaScript*-скриптов разработана и постоянно пополняется добровольцами библиотека *jQuery*, основная цель создания которой состоит в предоставлении возможностей создания много-разовых фрагментов кода *JavaScript*, позволяющих упростить процесс создания веб-страниц.

Методические рекомендации содержат 17 работ, в каждой из которых 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 работы 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*.

Выполнение каждой работы следует выполнять в следующем порядке:

- 1) ознакомиться с теоретическими положениями работы;
- 2) выбрать из таблицы «Варианты заданий» задание и выполнить его.

Отчет содержит: название и цель работы; постановку задачи; исходные данные; использованные технологии; результаты выполнения и анализ полученных результатов; выводы. В отчете можно привести также ответы на наиболее сложные вопросы, имеющиеся в конце каждой работы.

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

## Цель работы.

Ознакомиться с языком разметки гипертекста *html*.

## Методические материалы.

Для создания веб-страниц используется язык разметки гипертекста *html*, 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>` (нenumерованные), а их элементы – тегами `<li>`.

Для обозначения элементов нenumерованных 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; *l* – 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 `<div>`. Является блочным элементом и предназначен для выделения фрагмента документа с целью изменения вида содержимого. Как правило, вид блока управляется с помощью стилей.

Тег `<span>`. Предназначен для определения строчных элементов документа. В отличие от блочных элементов, таких как `<table>`, `<p>` или `<div>`, с помощью тега `<span>` 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 `<span>` 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, а имя файла – «*Ваше Имя*».html, вторая – «Элемент гиперссылки», указанный в колонке 7 таблицы 1.1.



Рисунок 1.1 – Вид *html*-страницы

### Листинг

```
<!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>

```

Таблица 1.1 – Варианты заданий

Номер варианта	Выравнивание «Текста страницы» по краю	Ширина левого, правого меню, %		Теги <i>h<sub>i</sub></i> заголовков меню	Элемент гиперссылки «Низ сайта»
		4	5		
1	2	4	5	6	7
1	Правому	18	17	h6	Отдых
2	По центру	15	20	h2	Погода
3	Нижнему	19	16	h4	Товары
4	Верхнему	20	17	h3	Услуги
5	Левому	14	18	h2	Работа
6	Верхнему	18	20	h5	Новости
7	По центру	17	15	h3	Отдых
8	Верхнему	14	17	h4	Погода
9	Правому	19	20	h5	Товары
10	Левому	20	19	h6	Услуги

2 С помощью тега *span* поменять цвет двух первых строк заголовка *h<sub>i</sub>* левого элемента «Меню» веб-страницы.

3 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.

### Контрольные вопросы

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

- 6 Как вставить рисунок в *html*-документ?
- 7 Из каких разделов состоит *html*-документ?

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

### Цель работы.

Ознакомиться с каскадной таблицей стилей *CSS*.

### Методические материалы.

*CSS* – это каскадные таблицы стилей, используемые для создания и изменения стиля элементов веб-страниц.

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

Селектор – это часть *CSS*-правила, которая сообщает браузеру, к какому элементу (или элементам) веб-страницы будет применён стиль.

Термин селектор может относиться к простому селектору (*simple selector*), составному (*compound selector*), сложному (*complex selector*) или списку селекторов.

К простым селекторам относятся: селектор типа, универсальный селектор, селекторы атрибутов, селектор идентификатора, селектор класса, псевдо-классы.

Составной селектор – это последовательность простых селекторов, которые не разделены комбинаторами, т. е. за одним селектором сразу идёт следующий. Он выбирает элемент, который соответствует всем простым селекторам, которые он содержит. Селектор типа или универсальный селектор, входящий в составной селектор, должен быть расположен первым в этой последовательности. В составном селекторе допустим только один селектор типа или универсальный селектор.

```
span[title].className
p.className1.className2#someId:hover
```

Сложный селектор – это последовательность селекторов, которые разделены комбинаторами.

Список селекторов – это селекторы, перечисленные через запятую.

Селектор – это тег *html*, сообщающий браузеру, какой элемент необходимо форматировать, а определение – его формирующие команды, задающие 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.

В таблице 2.1 приведены типы CSS-селекторов.

Таблица 2.1 – Типы CSS

1	2	3	4
<i>class</i>	<i>.myClass</i>	Выбирает все элементы с классом <i>myClass</i> ( <i>class = "myClass"</i> )	1
<i>#id</i>	<i>#main</i>	Выбирает элемент с идентификатором <i>main</i> ( <i>id = "main"</i> )	1
*	*	Выбор всех элементов	2
<i>элемент</i>	<i>span</i>	Выбор всех элементов <i>&lt;span&gt;</i>	1
<i>элемент, элемент</i>	<i>div, span</i>	Выбор всех элементов <i>&lt;div&gt;</i> и всех элементов <i>&lt;span&gt;</i>	1
<i>[атрибут]</i>	<i>[title]</i>	Выбирает все элементы с атрибутом <i>title</i>	2
<i>[атрибут="значение"]</i>	<i>[title = "cost"]</i>	Выбирает все элементы с атрибутом <i>title</i> , значения которых совпадают со значениями в селекторе ( <i>title = "cost"</i> ).	2
<i>[атрибут~="значение"]</i>	<i>[title~="один"]</i>	Выбирает все элементы с атрибутом <i>title</i> , в значении которого (в любом месте) встречается подстрока (в виде отдельного слова) «один» ( <i>title = "один и два"</i> )	2
<i>[атрибут = "значение"]</i>	<i>[lang  = "ru"]</i>	Выбор всех элементов с атрибутом <i>lang</i> , значение которого начинается с "ru"	2
<i>[атрибут^="значение"]</i>	<i>a[href^="https"]</i>	Выбор каждого элемента <i>&lt;a&gt;</i> с атрибутом <i>href</i> , значение которого начинается с "https"	3
<i>[атрибут\$="значение"]</i>	<i>[src\$ = ".png"]</i>	Выбирает все элементы с атрибутом <i>src</i> , значение которого оканчивается на ".png" ( <i>src = "some img.png"</i> )	3
<i>[атрибут*="значение"]</i>	<i>[title* = "один"]</i>	Выбирает все элементы с атрибутом <i>title</i> , в значении которого (в любом месте) встречается подстрока (в виде отдельного слова или его части) "один" ( <i>title = "один и два"</i> )	3

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 1 и 2.

Пункт – 1	Пункт – 2
Команда 1-1	Команда 2-1
Команда 1-2	Команда 2-2

Рисунок 2.1 – Вид меню

### Листинг 1 – html-код страницы

```

<!DOCTYPE html>
<html>
<head>
<title>Меню</title>
<link rel="stylesheet" type="text/css"
href="css/style.css" />
</head>
<body>
<div id="mainmenu">
  <ul id="nav">
    <li id="settings">
      <a href="#"><img
        <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>
  src="images/settings.png"> </a>
</li>
</body>
</html>

```

### Листинг 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 2–4 (см. таблицу 2.2). 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 столбце 4

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 17 px, слева – на 10 px и содержать слева графический элемент.

Таблица 2.2 – Варианты заданий

Номер варианта	Тема	Главная лента меню: фон/текст	Подменю 1: пунктов/фон/текст
1	2	3	4
1	Швейное ателье	Верт/ <i>Khaki/Aqua</i>	2/ <i>Purple/Navy</i>
2	Ремонт авто	Верт/ <i>Coral/Blue</i>	2/ <i>Navy/Navy</i>
3	Расписание занятий	Гориз/ <i>Gold/Navy</i>	3/ <i>Olive/Aqua</i>
4	Детские игрушки	Гориз/ <i>Pink/Red</i>	2/ <i>Aqua/Violet</i>
5	Ремонт компьютеров	Гориз/ <i>Plum/Black</i>	3/ <i>Black/Red</i>
6	Торговая фирма	Верт/ <i>Purple/Violet</i>	2/ <i>Tan/Gold</i>
7	Ремонт телевизоров	Гориз/ <i>Indigo/Tan</i>	2/ <i>Pink/Blue</i>
8	Головные уборы	Верт/ <i>Peru/Coral</i>	3/ <i>Blue/Blue</i>
9	Зимняя одежда	Гориз/ <i>Gray/Gold</i>	3/ <i>Blue/Red</i>
10	Продукты питания	Верт/ <i>Red/Plum</i>	2/ <i>Olive/Aqua</i>

### **Контрольные вопросы**

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

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

### **Цель работы.**

Ознакомиться с методом создания сайта, содержащего сценарии на языке *JavaScript*.

### **Методические материалы.**

Язык программирования *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:

1) в теговом контейнере `<body>...</body>`:

```
<body>
...
  <script > команды скрипта</script>
</body>
```

2) 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>
```

3) 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 3.1 номерами и разделить их номером варианта, например, для варианта 10: `a10c10e10` и т. д.

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 3.1 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.

Таблица 3.1 – Задания к лабораторной работе

Номер варианта	Задание 1 – номера символов	Задание 2 – операции		
1	2	3		
1	10, 13, 17, 19, 22, 25	+	–	*
2	02, 05, 07, 09, 12, 14	+	–	/
3	04, 05, 07, 09, 12, 18	+	–	%
4	13, 15, 17, 19, 21, 23	/	*	–
5	01, 04, 08, 12, 17, 21	/	*	+
6	03, 05, 06, 17, 18, 20	/	*	%
7	05, 07, 09, 13, 16, 18	*	+	–
8	05, 07, 09, 12, 14, 18	*	+	/
9	02, 09, 16, 17, 18, 21	*	+	%
10	08, 12, 18, 19, 20, 24	–	*	/

### **Контрольные вопросы**

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

### **Цель работы.**

Ознакомиться с операторами ветвления и циклов *JavaScript*.

### **Методические материалы.**

В сценариях на языке *JavaScript* для организации ветвлений используются конструкции *if*, *else*, *switch*, обеспечивающие выполнение определенной команды или набора команд только при условии истинности некоторого логического выражения:

```

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 16 до 0:

```

for (var i = 16; 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 16:

```
var i = 1;
while (i <= 16) {
    document.write(i + ' ');
    i++;
}
```

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

### Порядок выполнения работы.

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

1 Ввести переменную *lang*, которая может принимать значения: рус, англ, 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.

Варианты заданий приведены в таблице 4.1.

Таблица 4.1 – Варианты заданий

Номер варианта	Задача 1		Задача 2	
	<i>lang</i>	операт / массив	строка	<i>k</i>
1	2	3	4	5
1	англ	<i>If</i>	'5abcde12345'	11
2	рус	<i>switch-case</i>	'1ab123cde45'	7
3	бел	массив	'4abcd12345e'	2
4	англ	<i>If</i>	'3ab12cde345'	5
5	рус	<i>switch-case</i>	'7a12bcde345'	4
6	бел	массив	'4a1234bcde5'	9
7	англ	<i>If</i>	'r123abc45de'	3
8	бел	<i>switch-case</i>	's123abcde45'	5
9	рус	массив	'r12ab34cde5'	7
10	нем	<i>If</i>	'k123abc45de'	6

### Контрольные вопросы

- 1 Прокомментируйте назначение и структуру оператора *switch-case*.
- 2 Прокомментируйте синтаксис оператора *for*.



- 3 Для чего предназначена инструкция *break*?
- 4 Какие варианты использования оператора *if* Вы знаете?
- 5 Какие операторы цикла Вы знаете?

## 5 Лабораторная работа № 5. Изучение функций и методов JavaScript

### Цель работы.

Ознакомиться с функциями и методами *JavaScript*.

### Методические материалы.

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 ее 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* – декодирует *URI*, закодированный при помощи *encodeURI*;

*decodeURIComponent* – декодирует *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 12 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.

Таблица 5.1 – Варианты заданий

Номер варианта	Задание 1		Задание 2
	Функция	Сортировать	Даты
1	2	3	4
1	$\sin(x)$	По возрастанию	День, час, минута
2	$\cos(x)$	По убыванию	Месяц, день, час
3	$\text{tg}(x)$	По возрастанию	Час, минута, секунда
4	$\text{ctg}(x)$	По убыванию	Год, месяц, день
5	$\sin(x/2)$	По возрастанию	День, час
6	$\cos(x/2)$	По убыванию	Месяц, день, час
7	$\text{tg}(x/2)$	По возрастанию	Час, минута, секунда
8	$\text{ctg}(x/2)$	По убыванию	Минута, год, месяц, день
9	$\sin(x^2)$	По возрастанию	Год, месяц, день
10	$\cos(x^2)$	По убыванию	Месяц, день, час

### **Контрольные вопросы**

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

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

### **Цель работы.**

Ознакомиться с приемами работы с массивами *JavaScript*.

### **Методические материалы.**

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 6.1. Добавить  $m$  числовых элементов в его начало и  $n$  текстовых – в конец.

2 Создать двухмерный массив размерностью  $m \times n$  (столбец 4 таблицы 6.1) и заполнить случайными равномерно распределенными числами. Строки с четными номерами рассортировать.

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

### Контрольные вопросы

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

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

### Цель работы.

Ознакомиться с приемами работы с элементами управления *JavaScript*.

### Методические указания.

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*:

```



```

### 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.

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
1	2	3
1	45 %, 2 пикс	Пол, МР, ОБР
2	72 %, 3 пикс	ДР, пол, должность
3	85 %, 4 пикс	МР, возраст, пол
4	57 %, 2 пикс	МР, должность, пол
5	80 %, 3 пикс	Пол, должность, ДР
6	90 %, 2 пикс	НАЦ, пол, возраст
7	50 %, 2 пикс	СП, пол, должность
8	70 %, 3 пикс	Пол, НАЦ, возраст
9	80 %, 4 пикс	МР, возраст, должность
10	35 %, 2 пикс	ДР, должность, возраст

### Контрольные вопросы

- 1 Что такое динамически меняющаяся кнопка?
- 2 Как работают динамически изменяющиеся кнопки?
- 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 Для чего используется генератор всплывающих (*popup*-) окон?

## 8 Лабораторная работа № 8. Изучение основных методов JQuery

### Цель работы.

Ознакомиться с основными методами *JQuery*.

### Методические указания.

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



отобразить 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 <i>jQuery</i> 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)

Окончание таблицы 8.1

Номер варианта	Используя библиотеку <i>Jquery</i> создать сайт, содержащий
6	Перемещающиеся блоки
7	Изменение прозрачности элементов
8	Подсвечивание текста
9	Переливание цвета текста разными оттенками
10	Плавающее меню

### **Контрольные вопросы**

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

## **9 Лабораторная работа № 9. Изучение основных событий JQuery**

### **Цель работы.**

Ознакомиться с основными событиями *JQuery*.

### **Методические указания.**

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 отпуске 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 <i>JQuery</i> , создать сайт, содержащий
1	События мыши <i>.dblclick()</i> и <i>.mousemove()</i>
2	События документа/окна <i>.load()</i> и <i>.scroll()</i>
3	События мыши <i>.click()</i> и <i>.mousedown()</i>
4	События документа/окна <i>.resize()</i> и <i>.unload()</i>
5	События форм <i>.blur()</i> , <i>.focus()</i> и <i>.select()</i>
6	События форм <i>.change()</i> , <i>.reset()</i> и <i>.submit()</i>
7	События клавиатуры <i>.keydown()</i> и <i>.keyup()</i>
8	События клавиатуры <i>.keypress()</i> и <i>.keyup()</i>
9	Событие <i>jQuery.hover()</i>
10	Событие <i>jQuery.toggle()</i>

### Контрольные вопросы

- 1 Какие события мыши Вы знаете?
- 2 Что представляют собой события *JQuery*?
- 3 С помощью каких событий производится управление веб-страницей?
- 4 Какие события клавиатуры Вы знаете?
- 5 Какие события форм Вы знаете?

## 10 Лабораторная работа № 10. Установка локального сервера МАМР

### Цель работы.

Ознакомиться с установкой локального сервера *МАМР*.

### Методические указания.

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 др.

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.

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).

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



Рисунок 10.1 – Диалоговое окно *MAMP*

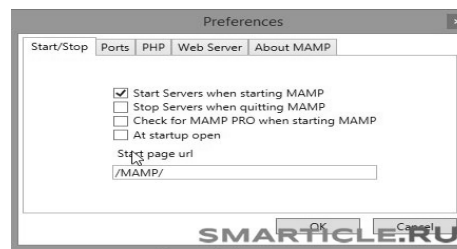


Рисунок 10.2 – Меню *Preferences*

После успешных настроек должны загореться два пункта зеленым цветом – *Apache* и *MySQL* (см. рисунок 10.1), 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).

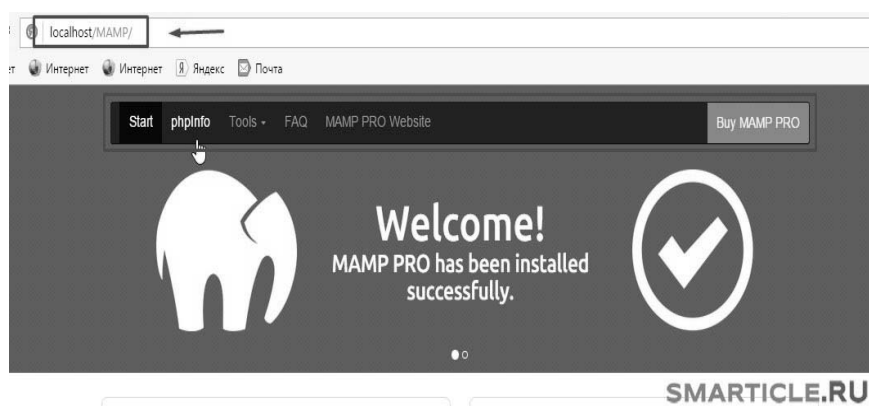


Рисунок 10.3 – Окно браузера

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 *CMS*?
- 2 Что такое локальный веб-сервер?
- 3 Для чего используется локальный веб-сервер *Apache*?
- 4 Как установить *CMS WordPress*?
- 5 Как создать статическую страницу в *WordPress*?

## 11 Лабораторная работа № 11. Изучение строковых функций языка PHP

### Цель работы.

Ознакомиться со строковыми функциями языка PHP.

### Методические указания.

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).

### 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 11.1 – Некоторые функции обработки строк

Номер функции	Функция
1	2
1	<i>chunk_split</i> – разбивает строку на фрагменты
2	<i>echo</i> – выводит одну или более строк
3	<i>explode</i> – разбивает строку с помощью разделителя
4	<i>implode</i> – объединяет элементы массива в строку
5	<i>lcfirst</i> – преобразует первый символ строки в нижний регистр
6	<i>ltrim</i> – удаляет пробелы или другие символы из начала строки

## Продолжение таблицы 11.1

1	2
7	<i>print</i> – выводит строку
8	<i>printf</i> – выводит отформатированную строку
9	<i>rtrim</i> – удаляет пробелы или другие символы из конца строки
10	<i>similar_text</i> – вычисляет степень похожести двух строк
11	<i>sprintf</i> – возвращает отформатированную строку
12	<i>sscanf</i> – разбирает строку в соответствии с заданным форматом
13	<i>strpos</i> – возвращает позицию первого вхождения подстроки без учета регистра
14	<i>strlen</i> – возвращает длину строки
15	<i>str_pad</i> – дополняет строку другой строкой до заданной длины
16	<i>str_replace</i> – заменяет все вхождения строки поиска на строку замены
17	<i>str_shuffle</i> – переставляет символы в строке случайным образом
18	<i>str_split</i> – преобразует строку в массив
19	<i>str_word_count</i> – возвращает информацию о словах, входящих в строку
20	<i>strip_tags</i> – удаляет теги <i>html</i> и PHP из строки
21	<i>strpbrk</i> – ищет в строке любой символ из заданного набора
22	<i>strpos</i> – возвращает позицию первого вхождения подстроки
23	<i>strrchr</i> – находит последнее вхождение символа в строке
24	<i>strrev</i> – переворачивает строку задом наперед
25	<i>stripos</i> – возвращает позицию первого вхождения подстроки без учета регистра
26	<i>strlen</i> – возвращает длину строки
27	<i>strpbrk</i> – ищет в строке любой символ из заданного набора
28	<i>strpos</i> – возвращает позицию первого вхождения подстроки
29	<i>strrchr</i> – находит последнее вхождение символа в строке
30	<i>strrev</i> – переворачивает строку задом наперед
31	<i>strripos</i> – возвращает позицию последнего вхождения подстроки без учета регистра
32	<i>strrpos</i> – возвращает позицию последнего вхождения подстроки в строке
33	<i>strspn</i> – возвращает длину участка в начале строки, соответствующего маске
34	<i>strstr</i> – находит первое вхождение подстроки
35	<i>strtok</i> – разбивает строку на токены
36	<i>strtolower</i> – преобразует строку в нижний регистр
37	<i>strtoupper</i> – преобразует строку в верхний регистр
38	<i>strtr</i> – преобразует заданные символы или заменяет подстроки
39	<i>substr_count</i> – возвращает число вхождений подстроки
40	<i>ltrim</i> – удаляет пробелы или другие символы из начала строки
41	<i>print</i> – выводит строку
42	<i>printf</i> – выводит отформатированную строку
43	<i>rtrim</i> – удаляет пробелы или другие символы из конца строки
44	<i>similar_text</i> – вычисляет степень похожести двух строк
45	<i>sprintf</i> – возвращает отформатированную строку
46	<i>sscanf</i> – разбирает строку в соответствии с заданным форматом
47	<i>strpos</i> – возвращает позицию первого вхождения подстроки без учета регистра
48	<i>strlen</i> – возвращает длину строки
49	<i>str_pad</i> – дополняет строку другой строкой до заданной длины
50	<i>str_replace</i> – заменяет все вхождения строки поиска на строку замены
51	<i>str_shuffle</i> – переставляет символы в строке случайным образом



Окончание таблицы 11.1

1	2
52	<i>str_split</i> – преобразует строку в массив
53	<i>substr_replace</i> – заменяет часть строки
54	<i>substr</i> – возвращает подстроку
55	<i>trim</i> – удаляет пробелы или другие символы из начала и конца строки
56	<i>ucfirst</i> – преобразует первый символ строки в верхний регистр
57	<i>ucwords</i> – преобразует в верхний регистр первый символ каждого слова строки
58	<i>vfprintf</i> – записывает отформатированную строку в поток
59	<i>vprintf</i> – выводит отформатированную строку
60	<i>vsprintf</i> – возвращает отформатированную строку
61	<i>wordwrap</i> – переносит строку по указанному количеству символов
62	<i>serialize()</i> – создает строковое представление текущего состояния массива или объекта
63	<i>unserialize()</i> – восстанавливает состояние массива / объекта из строки

Таблица 11.2 – Варианты заданий

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

### **Контрольные вопросы**

- 1 Что понимается под *heredoc*-синтаксисом?
- 2 Каковы особенности строк, записанных в одинарных кавычках?
- 3 В чем особенности строк, записанных в двойных кавычках?
- 4 Как преобразовать строку в массив?
- 5 Какая функция переворачивает строку задом наперед?

## **12 Лабораторная работа № 12. Изучение операторов цикла языка PHP**

### **Цель работы.**

Ознакомиться с операторами цикла языка *PHP*.

### **Методические указания.**

В языке *PHP* существует несколько конструкций, позволяющих выполнять повторяющиеся действия в зависимости от условия. Это циклы *while*, *do ...while*, *foreach* и *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 тех чисел от 1 до 100, которые делятся на 7
2	Создать строку из шести символов, состоящую из случайных чисел от 1 до 9
3	Дан массив с числами. Найти сумму квадратов его элементов
4	Найти сумму корней чисел от 1 до 15. Результат округлить до двух знаков в дробной части
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 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. 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'

### Контрольные вопросы

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

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

### Цель работы.

Ознакомиться с приемами работы с массивами на языке *PHP*.

### Методические указания.

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 *\$low* 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, содержит ли $k$ -я строка символ @. Если не содержит, то вставить этот символ в конец строки
2	В массиве строк удалить все <i>html</i> -теги, заключенные в скобки < >
3	В массиве из $n$ строк проверить, начинается ли каждая строка символом "*", а строки без "*" перенести в другой массив
4	В массив случайным образом поместить строки, содержащие «цитата дня». Для выбора строки из массива случайным образом можно использовать функции <i>Shuffle(array arr)</i> или <i>arrayrand(array arr, int num)</i>
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

### **Контрольные вопросы**

- 1 Какие конструкции используются для создания массивов?
- 2 Допускается ли хранение в одном массиве значений разных типов?
- 3 Что такое ассоциированный массив?
- 4 Для чего используется функция *Shuffle(array arr)*?
- 5 Прокомментируйте назначение функции *arrayrand(array arr, int num)*.

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

### **Цель работы.**

Ознакомиться с технологией работы с функциями языка *PHP*.

### **Методические указания.**

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 «*RHP*-поиск». Для просмотра подробного описания с примером конкретной *RHP*-функции необходимо указать ее имя в строке *RHP*-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 *RHP*, 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 *RHP* 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	Задание
1	2
1	Создать три переменные, присвоить им числовые значения и вывести на экран их произведение. Создать и вызвать пользовательскую функцию, принимающую два аргумента по ссылке и один по значению, которая должна присваивать переменным другие числовые значения
2	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
3	Создать пользовательскую функцию, которая принимает два аргумента и возвращает их произведение. Вызвать и создать функцию, передав ей в качестве аргументов два числа, и результат вывести на экран.

## Окончание таблицы 14.1

1	2
4	Создать пользовательскую функцию, принимающую аргументы в массив переменной длины и выводящую их на экран. Для доступа к элементам массива использовать цикл <i>foreach</i> . 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 <i>\$arr</i> 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 <i>'floor'</i> и <i>'ceil'</i> .
7	Дано целое двухзначное число. Создать функцию, определяющую делители этого числа, т. е. число, на которое оно делится без остатка. Сформировать массив делителей заданного числа
8	Задать режим строгой типизации, использовать инструкцию <i>declare(strict_types =1)</i> , 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 <i>declare(strict_types =1)</i> , затем создать пользовательскую функцию <i>my_func()</i> , которая будет принимать два целочисленных аргумента и возвращать их произведение. Создать переменную <i>\$count_apples</i> 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

**Контрольные вопросы**

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

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

### Цель работы.

Ознакомиться с технологией работы с файлами *PHP*.

### Методические указания.

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	Найти все файлы из папки <i>test</i> и вставить в начало каждого файла полный путь к нему. Текст файла должен остаться в нем и начинаться с новой строки
2	Вывести на экран имена всех папок из папки <i>test</i> и их подпапок. Уровень вложенности папок может быть любым
3	Вывести на экран содержимое всех файлов из папки <i>test</i> и ее подпапок. Уровень вложенности папок может быть любым
4	Дан массив со строками. Создать папку <i>test</i> , а в ней – папки, названиями которых служат элементы этого массива
5	Найти все файлы из папки <i>test</i> 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 <i>test</i> все файлы размером более 1 Мбайта
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
9	Вывести на экран название всех файлов с расширением <i>txt</i> из папки <i>test</i>
10	В папке <i>test</i> есть файлы и подпапки. Вывести на экран содержимое всех файлов, которые лежат непосредственно в папке <i>test</i>

### Контрольные вопросы

- 1 С помощью какой *PHP*-функции можно создать пустой файл?
- 2 С помощью какой инструкции можно включать файлы кода в *PHP*-документ?
- 3 Какая *PHP*-функция используется для открытия файла?
- 4 Какие параметры используются в *PHP*-функции *copy()*?
- 5 Какая *PHP*-функция используется для записи строки в файл?

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

### Цель работы.

Ознакомиться с операционной системой ОС *Linux*.

### Методические указания.

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 применением 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 Чем отличается вывод команд *ls -F* и *ls -la*?
- 2 Что такое «жесткая ссылка»?
- 3 Что такое «символическая ссылка»?
- 4 Как осуществить просмотр подкаталогов и их содержимого?
- 5 Как осуществить просмотр скрытых файлов в домашнем каталоге?

## **17 Лабораторная работа № 17. Изучение ОС Android и ее установка**

### **Цель работы.**

Ознакомиться с операционной системой *Android*.

### **Методические указания.**

*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.

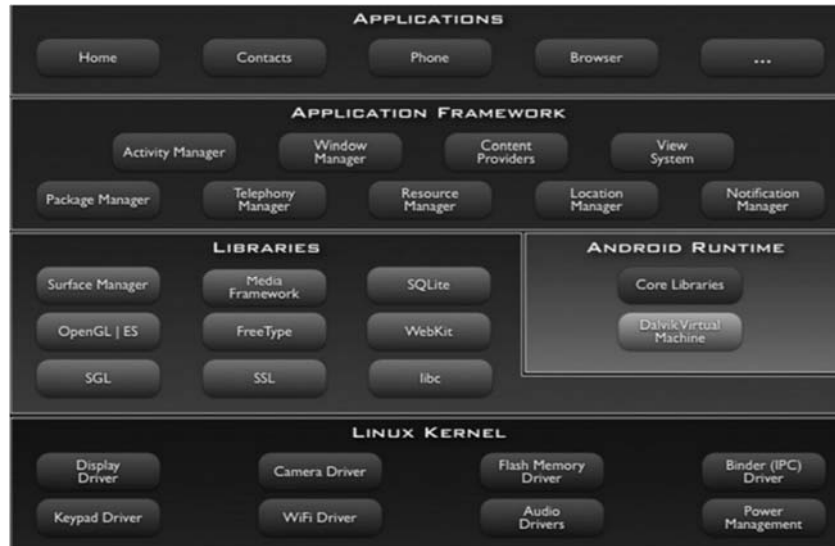


Рисунок 17.1 – Архитектура ОС *Android*

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*, используемая 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.

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 др.;

- 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 Гбайт, 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).

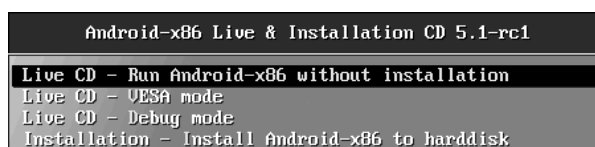


Рисунок 17.2 – Меню установки *Android*

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 Гбайт и отдельный дисковый раздел размером не менее 8 Гбайт. 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 Для чего смартфон на *Android* содержит ядро *Linux*?
- 2 Какие ОС используются в современных портативных устройствах?
- 3 Какие инструменты используются для разработки *Android*-приложений?
- 4 Возможна ли установка ОС *Android* на ПК, работающие под *Windows*?
- 5 Осуществляется ли загрузка ОС *Android* с флеш-накопителя?
- 6 Как создать дисковый раздел для установки *Android* на ПК?
- 7 Какие преимущества установки ОС *Android* в качестве второй операционной системы ПК?

### **Список литературы**

- 1 **Роббинс, Д.** HTML5, CSS3 и JavaScript. Исчерпывающее руководство (+ DVD-ROM) / Д. Роббинс. – Москва: Эксмо, 2018. – 528 с.
- 2 **Макфарланд, Д.** JavaScript и jQuery. Исчерпывающее руководство (+ DVD-ROM) / Д. Макфарланд. – Москва: Эксмо, 2017. – 688 с.
- 3 **Валади, Дж.** 100 % самоучитель Linux / Дж. Валади. – Москва: Технолоджи-3000, 2018. – 336 с.