

МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Автоматизированные системы управления»

СКРИПТОВЫЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ

*Методические указания к лабораторным
работам для студентов специальности*

1-40 05 01 Информационные системы и технологии (по направлениям)

1-40 05 01-1 Информационные системы и технологии (в проектировании и
производстве)

Могилёв 2022

Оглавление

Лабораторная работа №1	3
Лабораторная работа №2	38
Лабораторная работа №3	42
Лабораторная работа №4	48
Лабораторная работа №5	60
Лабораторная работа №6	68
Лабораторная работа №7	71
Лабораторная работа №8	86
Лабораторная работа №9	96
Лабораторная работа №10	116
Лабораторная работа №11	156
Лабораторная работа №13	171
Лабораторная работа №14	180
Лабораторная работа №15	194
Лабораторная работа №16	203
Лабораторная работа №17	214

Лабораторная работа №1

Создание HTML страницы с применением CSS

HTML – язык гипертекстовой разметки документов (HyperText Markup Language). С помощью HTML создаются Web-страницы, которые находятся в глобальной компьютерной сети Интернет. HTML – это не язык программирования в традиционном смысле, он является языком разметки. С помощью HTML текстовый документ разбивают на блоки смысловой информации (заголовки, параграфы, таблицы, рисунки и т.п.).

Гипертекстовый документ – это документ, содержащий переходы (**гиперссылки**) на другие документы. При использовании гиперссылки происходит перемещение от одного документа к другому (как по цепочке) в Интернете. HTML-документ является гипертекстовым документом.

Особенности HTML-документа:

1. HTML-документ может содержать текст, графику, видео и звук.
2. В общем случае HTML-документ – это один или несколько текстовых файлов, имеющих расширение .htm или .html.
3. Создавать HTML-документ можно как с помощью специальных программ – редакторов HTML (например, FrontPage), так и с помощью любого текстового редактора (например, блокнота Windows).
4. Для просмотра HTML-документов существуют специальные программы-**браузеры**. Они **интерпретируют** HTML-документы, т.е. переводят текст документа в Web-страницу, и отображают ее на экране пользователя. Существует очень много различных браузеров, но наиболее распространенными браузерами являются Microsoft Internet Explorer, Netscape Navigator и Opera.
5. Если при интерпретации HTML-документа браузер чего-то не понимает, то сообщения об ошибке не возникает, а это место в HTML-документе игнорируется и не отображается браузером.

HTML-документ состоит из элементов HTML.

Элемент HTML – это чаще всего два тега (открывающий и закрывающий) и часть документа между ними. Кроме того, существуют элементы HTML, состоящие только из одного тега.

Тег – в переводе с английского – ярлык, этикетка. Тег определяет тип выводимого элемента HTML (например, заголовок, таблица, рисунок и т.п.). Сам тег не отображается браузером. Тег представляет собой последовательность элементов:

- символ левой угловой скобки (<) – начало тега;
- необязательный символ слеша (/) – символ используется, чтобы обозначить закрывающий тег;
- имя тега;
- необязательные атрибуты в открывающем теге;

- символ правой угловой скобки (>)

Атрибуты – необязательный набор параметров, определяющих дополнительные свойства элемента HTML (например, цвет или размер). Атрибут состоит:

- из имени атрибута;
- знака равенства (=);
- значения атрибута – строки символов, заключенной в кавычки

Пример элемента HTML:

```
<H1 ALIGN= "CENTER">Глава 1</H1>
```

В этом примере:

<H1 ALIGN= "CENTER"> – открывающий тег

</H1> – закрывающий тег

H1 – имя тега

ALIGN= "CENTER" – атрибут

ALIGN – имя атрибута

"CENTER" – значение атрибута

Глава 1 – содержание элемента HTML

Правила создания HTML-документов:

1. Теги и атрибуты можно записывать в любом регистре, т.е. </H1> и </h1> – это одно и то же.
2. Несколько пробелов подряд, символы табуляции и перевода строки при интерпретации браузером заменяются на один пробел. Это позволяет писать хорошо структурированные исходные тексты файлов HTML.
3. Рекомендуется давать имена файлам HTML строчными английскими буквами. Длина имени – до восьми символов. В принципе, можно не придерживаться данной рекомендации, но тогда пользователи, работающие в операционных системах, отличных от Windows, не смогут воспользоваться вашими HTML-документами.

2. СТРУКТУРА HTML-ДОКУМЕНТА

Каждый HTML-документ должен начинаться тегом <HTML> и заканчиваться тегом </HTML>. Эти теги обозначают, что находящиеся между ними строки представляют единый HTML-документ. Кроме того, можно заметить, что файл HTML в целом является элементом языка HTML.

Также в HTML-документе должны присутствовать элементы HEAD (информация о документе) и BODY (тело документа).

2.1. РАЗДЕЛ ДОКУМЕНТА HEAD

Раздел документа HEAD определяет его заголовок, а также содержит дополнительную информацию о документе для браузера. Этот раздел не является обязательным, однако рекомендуется всегда использовать его в своих HTML-документах, так как правильно составленный заголовок может быть весьма полезен.

Раздел заголовка начинается тегом **<HEAD>** и следует сразу за тегом **<HTML>**. Между открывающим и закрывающим тегами элемента HEAD располагаются другие элементы заголовка.

Название документа TITLE

Для того чтобы дать название HTML-документу, предназначен тег **<TITLE>**. Это название будет выведено в заголовок окна браузера. Название записывается между тегами **<TITLE>** и **</TITLE>** и представляет собой строку текста. Длина этой строки может быть любой, но рекомендуется делать ее не больше 60 символов. Элемент TITLE должен находиться только в разделе HEAD.

2.2. РАЗДЕЛ ДОКУМЕНТА BODY

В этом разделе документа располагается та информация, которая отображается в окне браузера. Раздел BODY должен начинаться тегом **<BODY>** и завершаться тегом **</BODY>**, между которыми располагаются элементы HTML, из которых и состоит содержание документа.

2.2.1. Спецификация элемента BODY

Тег **<BODY>** имеет ряд атрибутов, определяющих внешний вид документа. Ниже приводится спецификация тега **<BODY>**.

```
<BODY  
  TEXT="цвет текста"  
  BGCOLOR="цвет фона"  
  BACKGROUND="адрес фонового рисунка"  
  LINK="цвет непосещенной гиперссылки"  
  VLINK="цвет посещенной гиперссылки"  
  ALINK="цвет активной гиперссылки"  
>
```

Атрибут TEXT задает цвет шрифта для всего документа в формате RGB или в символьной нотации. По умолчанию (если не указан этот атрибут) используются настройки браузера.

Атрибут BGCOLOR задает цвет фона окна браузера документа в формате RGB или в символьной нотации. По умолчанию используются настройки браузера.

Атрибут BACKGROUND позволяет указать адрес и имя рисунка, используемого в качестве фона. Этот рисунок будет размножен и распределен на заднем плане документа.

Атрибуты LINK, VLINK и ALINK задают цвета гиперссылок в формате RGB или в символьной нотации. По умолчанию используются настройки браузера. Непосещенная гиперссылка – гиперссылка, которая еще не использовалась для перехода к другому документу. Посещенная гиперссылка – гиперссылка, которая уже использовалась для перехода к другому документу. Активная гиперссылка – гиперссылка на документ, к которому в данный момент происходит переход.

2.2.2. Советы по использованию атрибутов тега BODY

- Если вы указываете хотя бы один цвет в теге BODY, то укажите и остальные. Это связано с тем, что пользователь может установить настройки цветов своего браузера как ему удобней. Указание только одного цвета может привести к ситуации, что у некоторых пользователей текст сольется с цветом фона. В результате просмотр документа будет затруднен.

- Выбирайте цвет текста так, чтобы он "работал" вместе с цветом фона или основными цветами изображения. Например, красное на зеленом может вызвать серьезные проблемы у значительного числа людей.

- В элементе BODY можно задать как BGCOLOR, так и BACKGROUND. В этом случае браузер отдает предпочтение BACKGROUND, но если изображение фона невозможно загрузить, будет использовано BGCOLOR. Поэтому старайтесь задавать цвет фона похожим на цвет фонового рисунка, чтобы не нарушился цветовой баланс документа.

2.3. ПРИМЕРЫ

2.3.1. Пример простого HTML-документа

```
<!DOCTYPE html>
<html>

<head>
  <title>Приветствие</title>
</head>
```

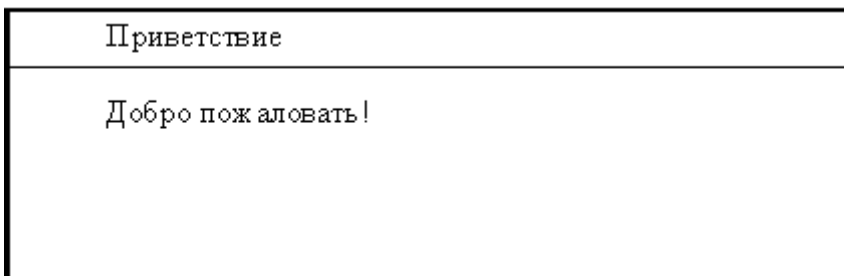
```

<body>
  <p>Добро пожаловать!</p>
</body>

</html>

```

Этот документ отобразится в браузере так:



В этом примере обратите внимание на то, куда выводится браузером название документа в элементе TITLE.

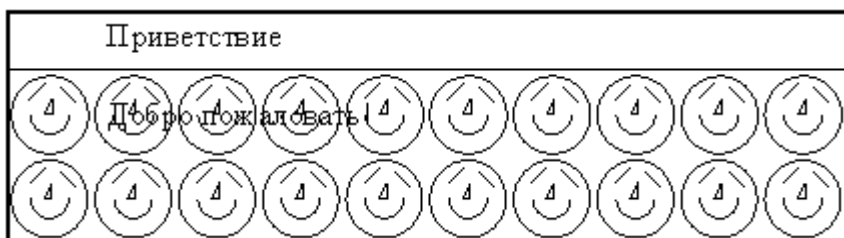
2.3.2. Пример использования фонового рисунка

```

<HTML>
<HEAD>
  <TITLE>Приветствие</TITLE>
</HEAD>
<BODY BACKGROUND="smail.gif">
  <P>Добро пожаловать!</P>
</BODY>
</HTML>

```

Этот документ отобразится в браузере так:



3. ЦВЕТОВЫЕ СПЕЦИФИКАЦИИ

Для определения цвета различных элементов HTML-документа необходимо указать значение соответствующих атрибутов. Указывать значения этих атрибутов можно двумя способами:

- определять цвет в формате RGB;
- определять цвет, используя символьную нотацию

3.1. ФОРМАТ RGB

Формат RGB – это система указания цвета, которая базируется на смешении трех основных цветов: красном (RED), зеленом (GREEN) и синем (BLUE). Итоговый цвет определяется цифрами в шестнадцатеричном коде. Для каждого цвета задается шестнадцатеричное значение в пределах от 0 до FF, что соответствует диапазону 0-255 в десятичном исчислении. Затем эти значения объединяются в одно число, перед которым ставится символ #. Например, число #800080 обозначает фиолетовый цвет. Указывая цвет в формате RGB, можно определить более шестнадцати миллионов цветовых оттенков.

3.2. СИМВОЛЬНАЯ НОТАЦИЯ

Задание цвета в формате RGB имеет один недостаток – необходимо помнить совокупности цифр для указания цвета. Этого недостатка лишена символьная нотация. Можно указывать название цвета на английском языке. Но у этого способа указания цвета тоже есть недостаток – определено всего шестнадцать имен цветов.

3.3. СООТВЕТСТВИЕ ФОРМАТА RGB И СИМВОЛЬНОЙ НОТАЦИИ

Ниже приведена таблица соответствий указания цвета в символьной нотации и формате RGB.

Символьная нотация	Формат RGB	Цвет
Black	#000000	Черный
Silver	#C0C0C0	Серебро
Gray	#808080	Серый
White	#FFFFFF	Белый
Maroon	#800000	Темно-бордовый
Red	#FF0000	Красный
Purple	#800080	Фиолетовый

Fuchsia	#FF00FF	Розовый
Green	#008000	Зеленый
Lime	#00FF00	Известь
Olive	#808000	Оливковый
Yellow	#FFFF00	Лимонный
Navy	#000080	Темно-синий
Blue	#0000FF	Синий
Teal	#008080	Темно-бирюзовый
Aqua	#00FFFF	Бирюзовый

Таким образом, строка `TEXT="#008000"` и строка `TEXT="Green"` в теге `<BODY>` одинаково определяют цвет шрифта – зеленый.

4. ВЫВОД ТЕКСТОВОЙ ИНФОРМАЦИИ

Любые тексты, будь то школьные сочинения, заметка в газету или техническое описание устройства, имеют определенную структуру. Элементами такой структуры являются заголовки, подзаголовки, абзацы, списки и др.

Разбиение всего текста на структурные элементы называется логическим форматированием. В HTML-документе логическое форматирование достигается с помощью специальных тегов.

4.1. АБЗАЦЫ

Одним из первых правил составления любых документов является разбиение его текста на отдельные абзацы, выражающие законченную мысль. В HTML-документе разделение на абзацы производится с помощью специального тега `<P>`. Синтаксис этого тега таков:

```
<P
  ALIGN="выравнивание">
```

Атрибут `ALIGN` определяет способ выравнивания абзаца. Он может иметь следующие значения:

- `LEFT` – текст выравнивается по левому краю окна браузера. Это значение используется по умолчанию, т.е. когда атрибут не указан.
- `CENTER` – текст выравнивается по центру окна браузера.
- `RIGHT` – текст выравнивается по правому краю окна браузера.

Пример использования тега `<P>`:

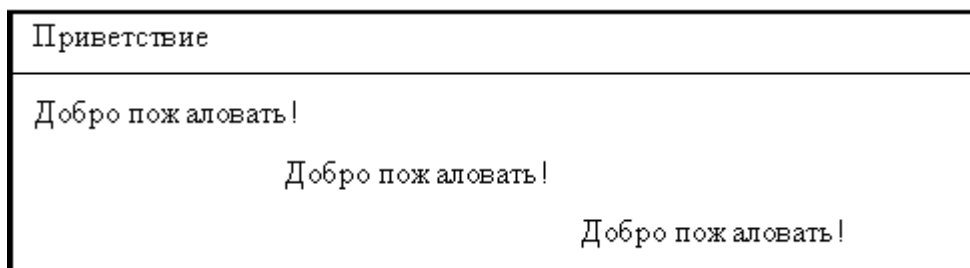
```
<HTML >
<HEAD >
```

```

<TITLE>Приветствие</TITLE>
</HEAD>
<BODY>
  <P>Добро пожаловать!</P>
  <P ALIGN="CENTER">Добро пожаловать!</P>
  <P ALIGN="RIGHT">Добро пожаловать!</P>
</BODY>
</HTML>

```

Этот документ отобразится в браузере так:



Браузер автоматически формирует абзацы в зависимости от ширины окна браузера или размера шрифта, перенося слова из строки в строку и отделяя абзацы друг от друга пустой строкой.

4.2. УПРАВЛЕНИЕ ПЕРЕВОДОМ СТРОКИ

Так как браузер автоматически определяет места переноса строк, иногда возникают ситуации запретить перевод строки в каком-нибудь месте или, наоборот, принудительно сделать перевод строки в каком-то определенном месте. Для этого существуют особые теги, управляющие переводом строк.

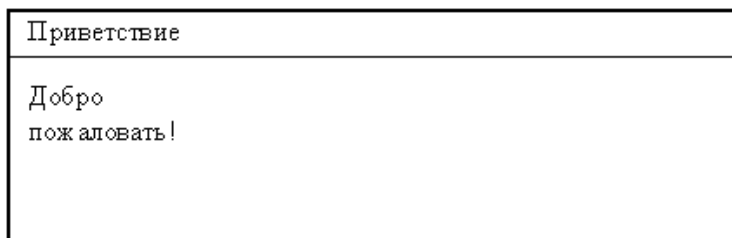
Когда необходимо сделать принудительный перевод строки, используют тег **
**. Этот тег не имеет атрибутов и закрывающего тега. Пример использования принудительного перевода строки:

```

<HTML>
<HEAD>
  <TITLE>Приветствие</TITLE>
</HEAD>
<BODY>
  <P>Добро<BR>позаловать!</P>
</BODY>
</HTML>

```

Этот пример будет выглядеть так:



При использовании тега `
` пустая строка не образуется, т.е. абзац не прерывается.

В некоторых случаях, наоборот, бывает необходимо сделать так, чтобы браузер не производил перевода строки. Например, не рекомендуется отрывать буквы инициалов от фамилии. В таких случаях тот участок текста, в котором нельзя переводить строку, следует поместить в элемент **NOBR**.

Пример:

`<P>`Это стихотворение написал `<NOBR>`А.С.
Пушкин`</NOBR>` – великий русский поэт.`</P>`

В браузере участок текста “А.С. Пушкин” всегда будет отображаться на одной строке.

Если получится так, что строка, расположенная в элементе **NOBR**, будет выходить за пределы окна браузера, то внизу окна появится горизонтальная полоса прокрутки.

4.3. ЗАГОЛОВКИ

Почти в каждом тексте используются заголовки для отдельных частей документа. Эти заголовки представляют собой фрагменты текста, которые выделяются на экране при отображении страницы браузером.

Для разметки заголовков используются теги `<H1>`, `<H2>`, `<H3>`, `<H4>`, `<H5>` и `<H6>`. Эти теги требуют соответствующего закрывающего тега. Заголовок с номером 1 является самым крупным (заголовок верхнего уровня), а с номером 6 – самым мелким. Теги заголовка нельзя использовать для выделения отдельных слов текста с целью увеличения их размера. При использовании тегов заголовков происходит вставка пустой строки до и после заголовка, поэтому тегов абзаца и перевода строки здесь не требуется.

Синтаксис тегов заголовков:

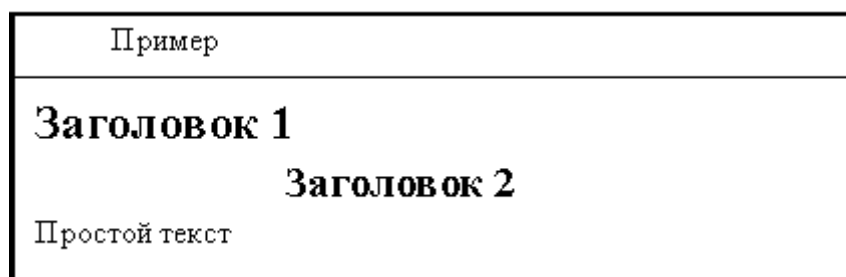
`<Hn`
`ALIGN="выравнивание">`

Атрибут **ALIGN** определяет способ выравнивания заголовка. Он может иметь те же значения, что и аналогичный атрибут у тега абзаца.

Пример использования разных заголовков:

```
<HTML>
<HEAD>
  <TITLE>Пример</TITLE>
</HEAD>
<BODY>
  <H1>Заголовок 1</H1>
  <H2 ALIGN="CENTER">Заголовок 2</H2>
  <P>Простой текст</P>
</BODY>
</HTML>
```

Вот так браузер отобразит данный пример:



4.4. ФОРМАТИРОВАНИЕ ТЕКСТА

В языке HTML предусмотрены специальные теги, предназначенные для форматирования текста. Они позволяют изменять вид шрифта, цвет, размер и др.

Чтобы отобразить текст полужирным шрифтом, используют тег ****.
Например:

```
<P>Это <B>полужирный</B> шрифт</P>
```

Тег **<I>** отображает текст курсивом. Например:

```
<P>Выделение <I>курсивом</I></P>
```

Используя тег **<TT>**, можно отобразить текст шрифтом, в котором все буквы имеют одинаковую ширину. Это так называемый моноширинный шрифт.
Например:

```
<P>Это <TT>моноширинный</TT> шрифт</P>
```

Для отображения текста подчеркнутым используется тег **<U>**. Например:

```
<P>Пример <U>подчеркивания</U> текста</P>
```

Тег **<S>** отображает текст, перечеркнутый горизонтальной линией.

Например:

<P>Пример <S>зачеркивания</S> текста</P>

Тег <BIG> выводит текст шрифтом большего (чем непомеченная часть текста) размера. Например:

<P>Шрифт <BIG>большого</BIG> размера</P>

Тег <SMALL> выводит текст шрифтом меньшего размера. Например:

<P>Шрифт <SMALL>меньшего</SMALL> размера</P>

Тег <SUB> сдвигает текст ниже уровня строки и выводит его (если возможно) шрифтом меньшего размера. Удобно использовать для математических индексов. Например:

<P>Шрифт _{нижнего} индекса</P>

Тег <SUP> сдвигает текст выше уровня строки и выводит его (если возможно) шрифтом меньшего размера. Удобно использовать для задания степеней чисел в математике. Например:

<P>Шрифт ^{верхнего} индекса</P>

Теги форматирования могут быть вложенными друг в друга.

При этом нужно внимательно следить, чтобы один контейнер находился целиком в другом контейнере. Например:

<P>Этот текст <I>полужирный и курсивный</I></P>

Вот так браузер отобразит приведенные выше примеры:

Форматирование текста
Это полужирный шрифт
Выделение <i>курсивом</i>
Это моноширинный шрифт
Пример <u>подчеркивания</u> текста
Пример зачеркивания текста
Шрифт БОЛЬШЕГО размера
Шрифт МЕНЬШЕГО размера
Шрифт _{нижнего} индекса
Шрифт ^{верхнего} индекса
Этот текст <i>полужирный и курсивный</i>

4.5.1. Тег

Тег позволяет изменить шрифт для блока текста. Этот тег имеет следующую спецификацию:

```
<FONT
  FACE="тип шрифта"
```

COLOR="цвет шрифта"
 SIZE="размер шрифта"

>

Атрибут FACE служит для указания типа шрифта, которым браузер будет выводить текст (если такой шрифт имеется на компьютере). Значением данного атрибута служит название шрифта, которое должно в точности совпадать с названием шрифта, имеющимся у пользователя. Если такой шрифт не найдется, то данное указание проигнорируется и будет использован шрифт, установленный по умолчанию.

Можно установить как один, так и несколько названий шрифтов, разделяя их запятыми. Тогда список шрифтов будет просматриваться слева направо так: если на компьютере пользователя нет шрифта, указанного в списке первым, то делается попытка найти следующий и т.д. Если ни одного шрифта найти не удалось, то будет использоваться шрифт, установленный браузером по умолчанию.

Атрибут COLOR устанавливает цвет шрифта. Значение этого атрибута может быть указано в формате RGB или символьной нотацией.

Атрибут SIZE служит для указания размера шрифта. Указывать размер шрифта можно двумя способами: абсолютной величиной или относительной величиной. При указании размера абсолютной величиной значением атрибута является число от 1 до 7. 1 – самый маленький шрифт, 7 – самый большой. При указании размера относительной величиной значением атрибута является число со знаком + или -. В данном случае шрифт будет увеличен (+) или уменьшен (-) от размера, принятого по умолчанию.

Пример использования тега :

```
<HTML>
```

```
<HEAD>
```

```
<TITLE>Пример изменения шрифта</TITLE>
```

```
</HEAD>
```

```
<BODY>
```

```
<P>Шрифт по умолчанию<BR>
```

```
<FONT COLOR="GREEN">
```

```
зеленый шрифт
```

```
</FONT><BR>
```

```
<FONT FACE="Arial">
```

```
другая форма шрифта
```

```
</FONT><BR>
```

```
<FONT SIZE="6">
```

```
размер шрифта – 6
```

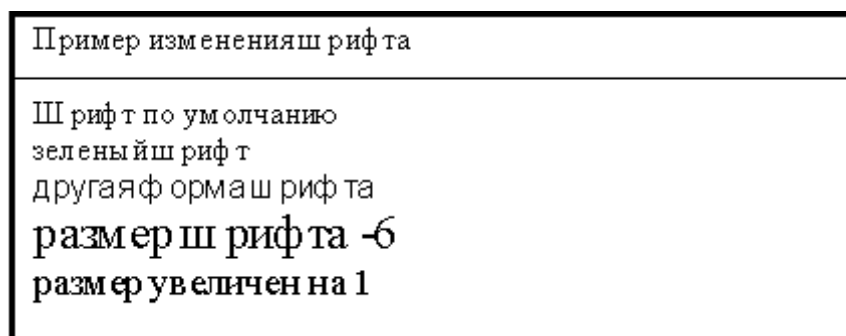
```

</FONT><BR>
<FONT SIZE="+1">
    размер увеличен на 1
</FONT>
</P>
</BODY>

</HTML>

```

Этот пример отобразится так:



4.5.2. Контейнер DIV

Иногда бывает необходимо произвести выравнивание большого блока документа, содержащего не только текст, но и рисунки, таблицы и т.п. Для этих целей используется элемент-контейнер **DIV**. Спецификация элемента DIV:

```

<DIV
    ALIGN="выравнивание">

```

Атрибут **ALIGN** определяет тип выравнивания содержимого и может иметь те же значения, что и элемент P.

4.5.3. Отступы

Иногда требуется отобразить блок текста с отступом. Для этого блок текста помещают в элемент-контейнер **BLOCKQUOTE**. Тогда содержимое этого элемента будет отображаться с небольшими отступами слева и справа, а также отделяться от остального текста пустыми строками.

5. ГИПЕРССЫЛКИ

Как уже было сказано, гиперссылки – переходы к другим документам. Они являются основой HTML. Гиперссылки можно использовать для перехода не только к другим HTML-документам, но и другим объектам, которые можно разместить на компьютере, например, к видео- и аудиофайлам, архивам, рисункам и т.п.

Каждая ссылка состоит из двух частей. Первая – это то, что отображается браузером. Она называется **указателем ссылки**. Вторая часть – адресная, содержащая адрес объекта, к которому должен происходить переход. Эту часть называют **URL** (универсальный идентификатор ресурсов).

Когда вы щелкаете мышью по указателю ссылки, браузер загружает документ, адрес которого указан в URL.

Указателем ссылки может быть слово, группа слов или рисунок. Если указатель текстовый, то он обычно отображается браузером подчеркнутым синим шрифтом. При наведении курсора мыши на указатель курсор принимает форму руки, указывая, что это ссылка и можно произвести переход. Если указатель графический, внешне он не отличается от других рисунков, но при наведении курсора мыши на такой указатель, курсор также принимает форму руки.

5.1. УНИВЕРСАЛЬНЫЙ ИДЕНТИФИКАТОР РЕСУРСОВ URL

По своей сути URL – это адрес файла, к которому происходит переход. Указание адреса может быть относительным или абсолютным.

Когда в URL указывается полный путь к файлу, независимо от того, где этот файл находится (в Интернете или на жестком диске компьютера), то это абсолютное указание. Например: **http://www.mysite.ru/page.htm** – абсолютный адрес документа, находящегося в Интернете, **c:\web\doc1.htm** – абсолютный адрес документа на диске **c**.

Если в URL указывается не полный путь, а путь относительно адреса документа, в котором находится ссылка, то это относительное указание. Например, браузер отображает документ, абсолютный адрес которого **c:\web\doc1.htm**, в этом документе имеется ссылка с адресом **pict/ris1.jpg**, то это означает, что на самом деле ссылка будет на документ по адресу **c:\web\pict\ris1.jpg**.

Когда надо сослаться на файл, находящийся на другом компьютере, тогда следует пользоваться абсолютным указанием, а если сослаться на файлы, находящиеся на том же компьютере, где и документ, содержащий ссылку, то лучше использовать относительное указание.

5.2. ПРАВИЛА ЗАПИСИ ССЫЛОК

Для организации ссылки необходимо указать браузеру, что является указателем ссылки, а также определить адрес документа, на который происходит ссылка. Оба этих действия выполняются с помощью тега <A>.

Тег <A> имеет следующую спецификацию:

```
<A
  HREF="URL-адрес"
  NAME="имя ссылки"
  TARGET="объект для вывода"
>
```

Атрибут HREF используется для задания адреса файла, к которому производится переход. Значением этого атрибута является текстовая строка, содержащая абсолютный или относительный URL-адрес.

Атрибут NAME предназначен для задания ссылке имени. Значением этого атрибута является короткая текстовая строка. Этот атрибут используется для ссылок внутри одного HTML-документа.

Атрибут TARGET позволяет определить, куда будет выводиться документ, на который происходит переход. Этот атрибут может иметь значение `_blank` – это означает, что документ будет выводиться в новом окне браузера.

Пример ссылки:

```
<A HREF="doc1.htm">Документ 1</A>
```

Браузер отобразит эту строку так:

Документ 1

При нажатии мышью на этой строке браузер загрузит и отобразит файл doc1.htm, т.е. “Документ 1” – это указатель ссылки, а “doc1.htm” – URL-адрес.

5.3. ВНУТРЕННИЕ ССЫЛКИ

Кроме ссылок на другие документы, часто бывает полезно включить ссылки на различные части текущего документа. Например, большой документ читается лучше, если он имеет оглавление со ссылками на соответствующие разделы.

Для построения внутренней ссылки сначала нужно создать указатель, определяющий место назначения. Для этого в месте, куда потом будет производиться ссылка, надо поместить тег <A> с атрибутом NAME, определив этим атрибутом имя указателя. Например:

```
<A NAME="glava5"></A>
```

Обратите внимание, что в этом примере отсутствует содержимое тега <A>. Обычно так и делают, поскольку здесь нет необходимости как-то выделять текст, а требуется лишь указать местоположение.

После того как место назначения определено, можно приступить к созданию ссылки на него. Для этого в атрибуте HREF тега <A> помещается

имя ссылки с префиксом #, говорящим о том, что это внутренняя ссылка. Например:

```
<A HREF="#glava5">Глава 5</A>
```

Теперь, если пользователь щелкнет кнопкой мыши на словах “Глава 5”, то браузер выведет соответствующую часть документа в окне просмотра.

Можно совмещать внутренние ссылки со ссылками на другие документы. Например:

```
<A HREF="doc1.htm#glava5">Глава 5 Документа 1</A>
```

При нажатии на эту ссылку браузер откроет файл doc1.htm, найдет в этом файле указатель glava5 и выведет в окне просмотра соответствующую информацию.

5.4. ССЫЛКИ НА ДОКУМЕНТЫ РАЗЛИЧНЫХ ТИПОВ

Когда пользователь щелкает мышью на ссылке, указывающей на другой HTML-документ, то документ выводится непосредственно в окне браузера. Если же ссылка указывает на документ иного типа, программа просмотра принимает документ и затем решает, что с ним делать дальше.

- Браузер знает этот тип документа и умеет с ним обращаться. Например, если создать ссылку на графический файл формата GIF и щелкнуть мышью на ней, то браузер очистит окно просмотра и отобразит указанное изображение.

- Браузер не распознает тип принятого документа и не знает, что с ним делать дальше. В этом случае он обратится к вспомогательным программам, имеющимся на компьютере пользователя. Если подходящая программа найдется, браузер запустит ее и передаст ей полученный документ для обработки. Например, если пользователь щелкнет на ссылке на видеофайл формата AVI, браузер загрузит файл, найдет программу для демонстрации AVI-файлов и запустит ее. Видеофайл будет показан в дополнительном небольшом окне.

5.5. ССЫЛКИ НА РЕСУРСЫ ИНТЕРНЕТА

Основное назначение HTML-документов – это глобальная компьютерная сеть Интернет. HTML-документ, размещенный в Интернете, становится Web-страницей. Чтобы обратиться к Web-странице, надо указать URL-адрес в такой форме: **http://sitename/docname**, где sitename – имя сайта, docname – имя документа. Например: <http://kotoz.newmail.ru/autor.htm>.

Можно на Web-странице разместить адрес электронной почты. Для этого URL-адрес указывается так: **mailto:address**, где address – это адрес почтового ящика. Например: <mailto:vasya@mail.ru>.

Также существуют специальные форматы URL-адреса для других ресурсов Интернета (FTP, TelNet, Newsgroup, Gopher, WAIS).

6. ГРАФИЧЕСКИЕ ЭЛЕМЕНТЫ

Одним из достоинств HTML-документа является возможность использования графических элементов в оформлении. Можно выделить три элемента, чаще всего используемых в HTML-документах: горизонтальные линии, таблицы и рисунки.

6.1. ГОРИЗОНТАЛЬНЫЕ ЛИНИИ

Горизонтальные линии визуально подчеркивают законченность той или иной области документа. Сейчас часто используют рельефную, вдавленную линию, чтобы обозначить “объемность” документа.

Тег **<HR>** позволяет провести рельефную горизонтальную линию в окне большинства браузеров. Этот тег не является контейнером, поэтому не требует закрывающего тега. До и после линии автоматически вставляется пустая строка. Спецификация тега **<HR>**:

```
<HR
  ALIGN="выравнивание"
  WIDTH="длина линии"
  SIZE="толщина линии"
  NOSHADE
>
```

Атрибут **ALIGN** определяет способ выравнивания линии. Он может иметь следующие значения:

- **LEFT** – линия выравнивается по левому краю окна браузера. Это значение используется по умолчанию.
- **CENTER** – линия выравнивается по центру окна браузера.
- **RIGHT** – линия выравнивается по правому краю окна браузера.

Атрибут **WIDTH** задает длину линии. Значением данного атрибута является число. Это число означает длину линии в пикселях. Если после числа стоит знак %, то это означает длину в процентах от ширины окна. Например:

<HR WIDTH="400"> – линия длиной 400 пикселей.

<HR WIDTH="50% "> – линия длиной 50 процентов от ширины окна.

Атрибут **SIZE** задает толщину линии. Значением этого атрибута является число. Это число означает толщину линии в пикселях.

Атрибут **NOSHADE** отменяет “трехмерность” линии.

6.2. РИСУНКИ

Без иллюстраций документ скучен, вял и однообразен. HTML позволяет использовать рисунки в формате JPG и GIF для оформления HTML-документов. Для вставки рисунков используется тег ****. Спецификация тега ****:

```
<IMG  
  SRC="адрес рисунка"  
  ALIGN="выравнивание"  
  HEIGHT="высота рисунка"  
  WIDTH="ширина рисунка"  
  BORDER="толщина рамки"  
>
```

Атрибут SRC определяет URL-адрес рисунка, который будет отображаться браузером.

Атрибут ALIGN определяет способ выравнивания рисунка. Он может иметь следующие значения:

- TOP – рисунок выравнивается по верхнему краю текущей строки.
- MIDDLE – рисунок выравнивается серединой по текущей строке.
- BOTTOM – рисунок выравнивается по нижнему краю текущей строки.
- LEFT – рисунок прижимается к левому краю окна браузера и обтекает текст.
- RIGHT – рисунок прижимается к правому краю окна браузера и обтекает текст.

Атрибут HEIGHT определяет высоту рисунка в пикселях.

Атрибут WIDTH определяет ширину рисунка в пикселях.

Используя атрибуты HEIGHT и WIDTH можно увеличивать или уменьшать рисунок. Если указать только один из этих атрибутов, то рисунок будет увеличен или уменьшен пропорционально и по ширине, и по высоте.

Атрибут BORDER позволяет задавать рамку вокруг рисунка. Значение этого атрибута – толщина рамки в пикселях. По умолчанию – 1.

Пример выравнивания рисунков:

```
<HTML>  
  
<HEAD>  
  <TITLE>Пример выравнивания</TITLE>  
</HEAD>  
  
<BODY>
```

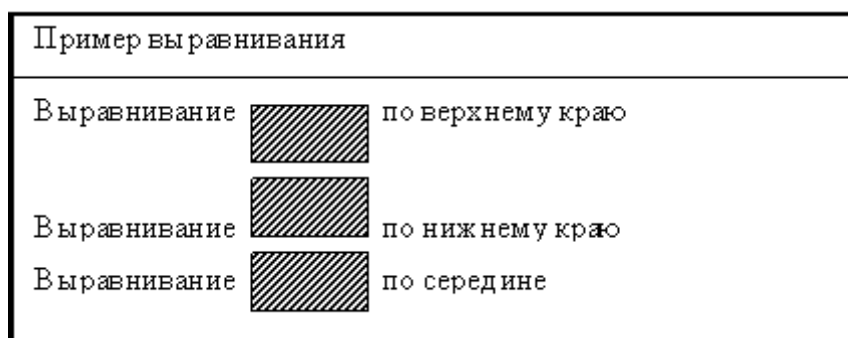
```

        <P>Выравнивание<IMG SRC="ris1.jpg"
ALIGN="TOP">по
        верхнему краю</P>
        <P>Выравнивание<IMG SRC="ris1.jpg"
ALIGN="BOTTOM">
        по нижнему краю</P>
        <P>Выравнивание<IMG SRC="ris1.jpg"
ALIGN="MIDDLE">
        по середине</P>
</BODY>

</HTML>

```

Браузер отобразит данный пример так:



6.3 РИСУНОК-ССЫЛКА

Можно использовать рисунки в качестве гиперссылок. Для этого нужно включить тег `IMG` в содержание элемента `A`. Например:

```
<A HREF="glava5.htm"><IMG SRC="ris1.jpg"></A>
```

7 СПИСКИ

В языке HTML предусмотрен специальный набор тегов для представления информации в виде списков. Списки являются одним из наиболее часто употребляемых форм представления данных как в электронных документах, так и печатных. В языке HTML предусмотрены маркированные, нумерованные списки и списки определений.

7.1. МАРКИРОВАННЫЙ СПИСОК

Этот список еще называется нумерованным или неупорядоченным. В маркированном списке для выделения его элементов используются специальные символы, называемые маркерами списка. Вид маркеров определяется браузером, причем при создании вложенных списков браузеры автоматически разнообразят вид маркеров различного уровня вложенности.

Для создания маркированного списка необходимо использовать тег-контейнер `` ``, внутри которого располагаются все элементы списка. Открывающий и закрывающий теги списка обеспечивают перевод строки до и после списка, отделяя, таким образом, список от основного содержимого документа, поэтому нет необходимости использовать теги абзаца или принудительного перевода строки.

Каждый элемент списка должен начинаться тегом `` и заканчиваться тегом ``.

Спецификация элемента UL:

```
<UL
  TYPE="вид маркера">
```

Атрибут TYPE задает вид маркера, которым выделяются элементы списка. Он может имеет следующие значения:

- DISK – маркеры отображаются закрашенными кружочками, это значение используется по умолчанию;
- CIRCLE – маркеры отображаются не закрашенными кружочками;
- SQUARE – маркеры отображаются квадратиками.

Спецификация элемента LI для маркированного списка:

```
<LI
  TYPE="вид маркера">
```

Атрибут TYPE задает вид маркера, он может принимать такие же значения, что и одноименный атрибут элемента UL. Значение по умолчанию – DISK.

Пример использования маркированного списка:

```
<HTML>

<HEAD>
  <TITLE>Пример списка</TITLE>
</HEAD>

<BODY>
  <UL>
    Крупные города России:
    <LI>Москва</LI>
```

```

    <LI>Санкт-Петербург</LI>
    <LI>Новосибирск</LI>
  </UL>
</BODY>

</HTML>

```

Вот так браузер отобразит данный пример:

Пример списка
<p>Крупные города России:</p> <ul style="list-style-type: none"> • Москва • Санкт-Петербург • Новосибирск

7.2. НУМЕРОВАННЫЙ СПИСОК

Нумерованные списки иногда называют упорядоченными. Списки данного типа представляют собой упорядоченную последовательность отдельных элементов. Отличием от маркированных списков является то, что в нумерованном списке перед каждым его элементом автоматически проставляется порядковый номер. Вид нумерации зависит от браузера и может задаваться атрибутами тегов списка. В остальном реализация нумерованного списка во многом похожа на реализацию маркированного списка.

Для создания нумерованного списка следует использовать тег-контейнер ` `, внутри которого располагаются все элементы списка.

Каждый элемент списка должен начинаться тегом `` и заканчиваться тегом ``.

Спецификация элемента OL:

```

<OL
  TYPE="вид нумерации"
  START="начальная позиция">

```

Атрибут TYPE задает вид нумерации, которой выделяются элементы списка. Он может иметь следующие значения:

- A – маркеры в виде прописных латинских букв;
- a – маркеры в виде строчных латинских букв;

- I – маркеры в виде больших римских цифр;
- i – маркеры в виде маленьких римских цифр;
- 1 – маркеры в виде арабских цифр, это значение используется по умолчанию.

Атрибут `START` определяет позицию, с которой начинается нумерация списка. Используя этот атрибут, можно начать нумерацию, например, с цифры 5 или буквы E, в зависимости от вида нумерации. Значением атрибута `START` является число, вне зависимости от вида нумерации.

Спецификация элемента `LI` для нумерованного списка:

```
<LI  
  TYPE="вид нумерации"  
  VALUE="номер элемента">
```

Атрибут `TYPE` задает вид нумерации, он может принимать такие же значения, что и одноименный атрибут элемента `OL`. По умолчанию значение этого атрибута – 1.

Атрибут `VALUE` позволяет изменить номер данного элемента, при этом изменятся номера и последующих элементов.

Пример использования нумерованного списка:

```
<HTML>  
  
<HEAD>  
  <TITLE>Пример списка</TITLE>  
</HEAD>  
  
<BODY>  
  <OL>  
    Города России по величине:  
    <LI>Москва</LI>  
    <LI>Санкт-Петербург</LI>  
    <LI>Новосибирск</LI>  
  </OL>  
</BODY>  
  
</HTML>
```

Вот так браузер отобразит данный пример:

Пример списка
<p>Города России по величине:</p> <ol style="list-style-type: none"> 1. Москва 2. Санкт-Петербург 3. Новосибирск

7.3. СПИСОК ОПРЕДЕЛЕНИЙ

Списки определений, также называемые словарями определений специальных терминов, являются особым видом списков. В отличие от других типов списков каждый элемент списка определений всегда состоит из двух частей. В первой части элемента списка указывается определяемый термин, а во второй части – текст в форме словарной статьи, раскрывающий значение термина.

Списки определений задаются с помощью тега-контейнера **<DL>**. Внутри него тегом **<DT>** отмечается определяемый термин, а тегом **<DD>** – абзац с его определением. Внутри элемента **<DT>** нельзя использовать абзацы (P) и заголовки (H1-H6), но их можно использовать внутри элемента **<DD>**. Атрибутов для элементов списка определений нет. В общем случае список определений записывается следующим образом:

```
<DL>
  <DT>Термин</DT>
  <DD>Определение термина</DD>
</DL>
```

Пример использования списка определений:

```
<HTML>

<HEAD>
  <TITLE>Пример списка определений</TITLE>
</HEAD>

<BODY>
  <DL>
    <H2 ALIGN="CENTER">Состав Microsoft
Office</H2>
```

```

        <DT>Microsoft Word</DT>
        <DD>Многофункциональный текстовый
процессор</DD>
        <DT>Microsoft Excel</DT>
        <DD>Программа для работы с электронными
таблицами</DD>
        <DT>Microsoft Access</DT>
        <DD>Система управления базами данных</DD>
    </DL>
</BODY>

</HTML>

```

Этот пример отобразится в браузере так:

Пример списка определений
Состав Microsoft Office
Microsoft Word Многофункциональный текстовый процессор
Microsoft Excel Программа для работы с электронными таблицами
Microsoft Access Система управления базами данных

8 ТАБЛИЦЫ

Одним из наиболее мощных и широко применяемых в HTML средств являются таблицы. Они используются не только традиционно как метод представления данных, но и как средство форматирования Web-страниц. Документ HTML может содержать произвольное число таблиц, причем допускается вложенность таблиц друг в друга.

Каждая таблица начинается тегом **<TABLE>** и заканчивается тегом **</TABLE>**. Внутри этой пары тегов располагается описание содержимого таблицы. Любая таблица состоит из одной или нескольких строк, в которых задаются данные для отдельных ячеек.

Каждая строка начинается тегом **<TR>** и заканчивается тегом **</TR>**. Отдельная ячейка в строке обрамляется парой тегов **<TD>** и **</TD>** или **<TH>**

и `</TH>`. Тег `<TH>` используется для ячеек заголовка таблицы, а `<TD>` – для ячеек данных. Отличие этих тегов в том, что в заголовке по умолчанию используется полужирный шрифт, а для данных – обычный.

Теги `<TD>` и `<TH>` не могут появляться вне описания строки таблицы `<TR>`.

Пример таблицы:

```
<HTML >

<HEAD>
  <TITLE>Пример таблицы</TITLE>
</HEAD>

<BODY>
  <TABLE >
    <TR>
      <TD>Ячейка 1</TD>
      <TD>Ячейка 2</TD>
    </TR>
    <TR>
      <TD>Ячейка 3</TD>
      <TD>Ячейка 4</TD>
    </TR>
  </TABLE >
</BODY >

</HTML >
```

Этот пример отобразится в браузере так:

Пример таблицы	
Ячейка 1	Ячейка 2
Ячейка 3	Ячейка 4

Спецификация тега `<TABLE>`:

`<TABLE`

`ALIGN="выравнивание"`

`BORDER="толщина рамки"`

`CELLPADDING="расстояние"`

```
CELLSPACING="расстояние"
HEIGHT="высота"
VALIGN="вертикальное выравнивание"
WIDTH="ширина"
```

>

Атрибут ALIGN определяет выравнивание таблицы в окне просмотра браузера. Он может иметь одно из двух значений: LEFT (по левому краю) и RIGHT (по правому краю). По умолчанию – LEFT.

Атрибут BORDER управляет толщиной рамки. Значением этого атрибута является число. Это число определяет толщину рамки таблицы в пикселях. По умолчанию толщина рамки – 1.

Атрибут CELLPADDING определяет расстояние в пикселях между рамкой и содержимым ячейки. По умолчанию – 1.

Атрибут CELLSPACING определяет расстояние в пикселях между ячейками таблицы. По умолчанию – 2.

Атрибут HEIGHT определяет высоту таблицы в пикселях.

Атрибут VALIGN определяет вертикальное выравнивание содержимого таблицы. Он может иметь следующие значения: TOP (по верхнему краю), MIDDLE (посередине) и BOTTOM (по нижнему краю). По умолчанию – MIDDLE.

Атрибут WIDTH определяет ширину таблицы в пикселях или процентах от ширины окна браузера.

Спецификация тега <TR>

<TR

```
ALIGN="выравнивание"
BGCOLOR="цвет фона"
VALIGN="вертикальное выравнивание"
```

>

Атрибут ALIGN определяет выравнивание содержимого всех ячеек строки. Он может иметь одно из трех значений: LEFT (по левому краю), RIGHT (по правому краю) и CENTER (по центру). По умолчанию – CENTER.

Атрибут BGCOLOR определяет цвет фона для всех ячеек строки. Его значение можно указывать в символьной нотации или в формате RGB.

Атрибут VALIGN определяет вертикальное выравнивание содержимого всех ячеек строки. Он может иметь следующие значения: TOP (по верхнему краю), MIDDLE (посередине) и BOTTOM (по нижнему краю). По умолчанию – MIDDLE.

Спецификация тега <TD>

<TD

```
ALIGN="выравнивание"
```

BGCOLOR="цвет фона"
 COLSPAN="количество ячеек"
 HEIGHT="высота ячейки"
 ROWSPAN=" количество ячеек "
 VALIGN="вертикальное выравнивание"
 WIDTH="ширина ячейки"

>

Атрибут ALIGN определяет выравнивание содержимого ячейки. Он может иметь одно из трех значений: LEFT (по левому краю), RIGHT (по правому краю) и CENTER (по центру). По умолчанию – CENTER.

Атрибут BGCOLOR определяет цвет фона для ячейки. Его значение можно указывать в символьной нотации или в формате RGB.

Атрибут COLSPAN позволяет объединить несколько соседних ячеек по горизонтали. Значение этого атрибута – количество объединяемых ячеек.

Атрибут HEIGHT определяет высоту ячейки в пикселях.

Атрибут ROWSPAN позволяет объединить несколько соседних ячеек по вертикали. Значение этого атрибута – количество объединяемых ячеек.

Атрибут VALIGN определяет вертикальное выравнивание содержимого ячейки. Он может иметь следующие значения: TOP (по верхнему краю), MIDDLE (посередине) и BOTTOM (по нижнему краю). По умолчанию – MIDDLE.

Атрибут WIDTH определяет ширину ячейки в пикселях.

9 СЛОИ

Слой представляет собой прямоугольный контейнер, в котором могут содержаться текст, рисунки, формы, подключаемые модули (plugins) и даже другие слои, т.е. любые элементы, которые можно поместить в тело HTML-документа. Слои обеспечивают разработчикам web-страниц полный контроль (вплоть до одного пикселя) над размещением элементов на странице.

Единственное, что нельзя сделать, используя слои, - это заставить слои отображаться одинаково в разных браузерах. Дело в том, что в Internet Explorer для описания слоя используется тег-контейнер <div> (иногда), а фирма Netscape для тех же целей изобрела свои собственные теги, не входящие в официальную спецификацию HTML, - <layer> и <ilayer>.

10. КАСКАДНЫЕ (МНОГОУРОВНЕВЫЕ) ТАБЛИЦЫ СТИЛЕЙ

Каскадные (многоуровневые) таблицы стилей это набор правил форматирования тегов HTML. Они дают возможность хранить содержимое отдельно от его представления.

Стиль включает все типы элементов дизайна: шрифт, фон, текст, цвета ссылок, поля и расположение объектов на странице.

CSS предполагает 3 типа таблиц стилей - встроенные, внедренные (внутренние) и связанные (внешние).

Существует три метода для применения таблицы стилей к документу HTML:

- Встроенный (Inline). Этот метод позволяет взять любой тег HTML и добавить к нему стиль. Использование встроенного метода предоставляет максимальный контроль над всеми свойствами Web-страницы. Предположим, вы хотите задать внешний вид отдельного абзаца. Вы можете просто добавить атрибут style к тегу абзаца, и браузер отобразит этот абзац с помощью параметров стиля, добавленного в код.

- Внедренный (Embedded). Внедрение позволяет контролировать всю страницу HTML. При использовании тега <style>, помещенного внутри раздела <head> страницы HTML, в код вставляются детализированные атрибуты стиля, которые будут применяться ко всей странице.

- Связанный (Linked или External). Связанная таблица стилей - мощный инструмент, который позволяет создавать образцы стилей (master styles), которые можно затем применять ко всему узлу. Основным документом таблицы стилей (расширение .css) создается Web-дизайнером. Этот документ содержит стили, которые будут едиными для всего Web-узла (неважно, содержит одну страницу или тысячи страниц). Любая страница, связанная с этим документом, будет использовать указанные стили.

Определение правил CSS

Правила определения CSS состоят из селекторов и описаний их свойств. В качестве селекторов используются стандартные имена тегов, свойства которых необходимо дополнить или изменить. Все, что находится в фигурных скобках принято называть определением (или описанием):

селектор {свойство 1: значение 1; свойство 2: значение 2;...}

Например, для того, чтобы цвет фона Web-страницы сделать черным, необходимо объявить следующее правило форматирования:

```
body{background:black}
```

В данном случае объявлено правило форматирования тега body, а именно - свойству стиля background присвоено значение black (черный). В результате применения этого правила цвет фона всего документа изменится на черный.

Изменим с помощью CSS не только цвет фона Web-страницы, но и цвет шрифта (на белый).

```
body{background:black;color:white}
```

Формат самого правила не имеет значения, главное - правило должно читаться удобно и легко. Например, вышеприведенное правило можно записать и так:

```
body{
background:black;
color:white}
```

Одно и то же правило стиля можно применить сразу к нескольким различным тегам HTML-страницы. Например:

```
body,td,h1{
background:black;
color:white}
```

Порой требуется одному и тому же тегу (элементу) придать разные свойства. Например, в одном случае заголовок типа h2 должен быть выделен зеленым цветом, а в другом этот же тип заголовка - красным. Подобные задачи решаются путем использования классов и идентификаторов. Во внешнем файле следует поместить описание двух классов:

```
h2.green{
    color : green
}
h2.red{
    color : red
}
```

Вызов созданных классов в местах форматирования текста осуществляется так:

```
<h2 class="green">Заголовок будет зеленым</h2>
<h2 class="red">Заголовок будет красным</h2>
```

Встроенный стиль

Встроенный стиль применяется к любому тегу HTML с помощью атрибута style следующим образом:

```
<P style="font: 12pt Courier New"> The text in this line will
display as 12 point text using the Courier New font.
</P>
```

Или:

```
<p style="font: 12pt Arial">
The text in this line will display as 18 point text using the
Arial font.
</p>
```

Можно добавлять встроенный стиль в любой тег HTML, в котором эта операция будет иметь смысл. Среди таких тегов можно назвать абзацы, заголовки, горизонтальные линии, якоря и ячейки таблиц. Ко всем этим элементам логично применять встроенные стили.

Существуют два тега, которые помогают применять встроенные стили к разделам страницы. Это теги `<div>` (division - раздел) и `` (промежуток).

Эти теги определяют диапазон текста, так что все, находящееся между ними, будет оформлено с помощью нужного стиля. Единственное различие между `<div>` и `` состоит в том, что `<div>` создает принудительный разрыв строки, а `` - нет.

Следовательно, нужно использовать `` для изменения стиля любой части текста, меньшей абзаца.

Вот пример работы тега `<div>`:

```
<div style="font-family: Garamond; font-size: 18 pt;">All of the
text within this section is 18 point Garamond.
```

All of the text within this section is 18 point Garamond.

и тега ``:

```
<span style="color:#ff3300;"> This text appears in the color red,
with no line break after the closing span tag </span> and the rest of
the text.
```

Внедренный стиль

Внедренные стили используют тег `<style>`, расположенный между тегами `</head>` и `<body>` в стандартном документе HTML.

Рассмотрим пример внедренного стиля:

```
<html>
<head>
  <title>Embedded Style Sheet Example </title>
</head>
<style>
  BODY {
    background: #FFFFFF;
    color: #000000;
  }

  H1 {
    font: 14pt verdana;
    color: #CCCCCC;
  }

  P {
    font: 12pt times;
  }

  A {
```



```

        color: #FF0000;
        text-decoration: none;
    }
</style>

```

Связанные таблицы стилей

Связанные (linked), или внешние (external), таблицы стилей являются расширением понятия внедренных стилей. Используется тот же самый код, что и для внедренной таблицы стилей, но он помещается в отдельный документ (файл с расширением .css). После этого с этим документом (файлом) связываются все страницы, к которым необходимо применить данный стиль.

Вот как выглядит пример связанной таблицы стилей:

```

<style>
BODY{
    background: #ffffcc;
    color: #000000;
}
P{
    font-family : sans-serif;
    font-style : italic;
    font-size : 16pt;
    color: #006633;
}
H1{
    font-family: Arial, Helvetica, sans-serif;
    font-size: 24pt;
    color: #996633;
}
</style>

```

Теперь сохраним этот документ как отдельный файл. Назовем его style-1.css и поместим в папку таблиц стилей с именем style.

С этим документом любое количество страниц HTML. Для этого нужно использовать между тегами </title> и </head> следующую конструкцию:

```
<link rel=stylesheet href="style-1.css" type="text/css">
```

Любая страница, содержащая такую связь, будет оформлена в соответствии со стилями, указанными в файле style_1.css.

Код этой страницы выглядит следующим образом:

```
<html lang="en">
```

```
<head>
  <title>Пример использования связанной таблицы
  стилей</title>
  <link href="style/style-1.css" rel=stylesheet
  type="text/css">
</head>

<body>
  <h1>Старинная студенческая песня</h1>
  <P>
    Поднявши меч на наш союз <br>
    достоин будет худшей кары,<br>
    и я за жизнь его тогда<br>
    не дам и ломанной гитары.<br>
    Как вожделенно жаждет век<br>
    нащупать брешь у нас в цепочке...<br>
    Возьмемся за руки друзья,<br>
    чтоб не пропасть поодиночке.<br>
  </P>

  <a href="css.htm#метка_51">Вернемся?</a>

</body>
</html>
```

Здесь важно помнить о смысле концепции каскадирования. Если вам нужны 10 страниц HTML, на которые глобально воздействует эта таблица стилей, вы можете ее применить. Затем, если нужно внести небольшие корректировки в отдельные страницы, можно либо внедрить в эти страницы дополнительные стили, либо использовать встроенный стиль.

Задания к лабораторной работе

Разработать три взаимосвязанных HTML-страницы для размещения на сайте. Первый HTML-документ предназначен для размещения на сайте в качестве первой страницы (имеющей обычно имя index.html, default.html или home.html). На странице должны располагаться следующие элементы:

- название фирмы;
- логотип фирмы;
- обращение к посетителю страницы;
- адрес фирмы;
- HTML-ссылки на две другие страницы сайта.

Второй HTML-документ предназначен для размещения простейшей рекламы товаров. На странице должны располагаться следующие элементы:

- название фирмы;
- логотип фирмы;
- обращение к посетителю страницы;
- прайс-лист в форме таблицы;
- HTML-ссылки на две другие страницы сайта.

Третий HTML-документ предназначен для размещения перечня услуг/товаров фирмы. На странице должны располагаться следующие элементы:

- заголовок “Перечень услуг/товаров” с указанием имени или фамилии учащегося;
- логотип фирмы;
- перечень;
- HTML-ссылки на две другие страницы сайта.

При разработке HTML-страницы использовать указанную CSS-таблицу.

Альтернативный вариант: разработка одностраничного лендинга (согласовать с преподавателем).

Вариант1

Название фирмы - заголовок пятого уровня, выравнивается по центру краю, буквы желтого цвета.

Остальной текст выравнивается по ширине, шрифт фиолетового цвета, начертание - полужирный, размер шрифта на 2 единицы больше текущего.

Таблица выровнена по правому краю, ширины рамки 3 пикселя, расстояние между границами ячеек равно 3, цвет заголовков - аквамарин.

Список нумерованный, нумерация арабскими цифрами.

Для задания свойств списка использовать внедренную таблицу стилей

Вариант2

Название фирмы - заголовок первого уровня, выравнивается по центру, буквы красного цвета.

Остальной текст по правому краю, шрифт синего цвета, начертание - курсив, размер шрифта на 2 единицы больше текущего.

Таблица выровнена по правому краю, ширины рамки 3 пикселя, расстояние между границами ячеек равно 4.

Список маркированный, маркеры – незакрашенные кружки.

Для задания свойств таблицы использовать внедренную таблицу стилей

Вариант3

Название фирмы - заголовок первого уровня, выравнивается по правому краю, буквы красного цвета.

Остальной текст выравнивается по правому краю, шрифт зеленого цвета, начертание - подчеркнутый, размер шрифта максимально возможный.

Таблица расположена по центру, с заголовком, расположенным над таблицей, ширины рамки 2 пикселя, ячейки, в которых расположен заголовок, голубого цвета.

Список многоуровневый. Тип маркеров каждого уровня выбрать самостоятельно.

Для задания свойств шрифта основного текста использовать внешнюю таблицу стилей.

Вариант4

Название фирмы - заголовок третьего уровня, выравнивается по правому краю, буквы желтого цвета.

Остальной текст выравнивается по центру, шрифт зеленого цвета, начертание - курсив, размер шрифта на 1 единицу больше текущего.

Таблица расположена по центру, с заголовком, расположенным над таблицей, ширины рамки 2 пикселя, ячейки, в которых расположен заголовок, желтого цвета.

Список маркированный, маркеры – закрашенные квадратики.

Для задания свойств списка использовать внешнюю таблицу стилей

Вариант5

Название фирмы - заголовок первого уровня, выравнивается по левому краю, буквы зеленого цвета.

Остальной текст по правому краю, шрифт синего цвета, начертание - курсив, размер шрифта на 2 единицы больше текущего. Таблица расположена по центру, с заголовком, расположенным над таблицей, ширины рамки 2 пикселя, ячейки, в которых расположен заголовок, серого цвета.

Список маркированный, маркеры – закрашенные кружки.
Для задания свойств таблицы использовать внешнюю таблицу стилей.

Лабораторная работа №2

Создание простых скриптов на JavaScript

Программа (сценарий) на языке JavaScript представляет собой последовательность инструкций (Инструкции – это синтаксические конструкции и команды, которые выполняют действия). Инструкции могут отделяться точкой с запятой. Обычно каждую инструкцию пишут на новой строке, чтобы код было легче читать. В большинстве случаев точку с запятой можно не ставить, если есть переход на новую строку. В этом случае JavaScript интерпретирует перенос строки как «неявную» точку с запятой. Это называется автоматическая вставка точки с запятой. В некоторых ситуациях новая строка всё же не означает точку с запятой.

Например:

```
alert(3 +  
1  
+ 2);
```

Но есть ситуации, где JavaScript «забывает» вставить точку с запятой там, где она нужна. Например, JavaScript не вставляет точку с запятой перед квадратными скобками [...]. Рекомендуется ставить точку с запятой между инструкциями, даже если они отделены переносами строк.

В программах на JavaScript можно использовать комментарии. Для того чтобы задать комментарий, располагающийся на одной строке, достаточно перед его текстом поставить две косые черты (//). Если же поясняющий текст занимает несколько строк, то его следует заключать между символами /* и */. В JavaScript строчные и прописные буквы алфавита считаются разными символами. Любой язык программирования оперирует с постоянными и переменными величинами. В JavaScript это литералы и переменные.

Опр.: Простейшие данные, с которыми может оперировать программа, называются литералами.

Литералы не могут изменяться. Литералы целого типа могут быть заданы в десятичном (по основанию 10), шестнадцатеричном (по основанию 16) или восьмеричном (по основанию 8) представлении. Шестнадцатеричные числа включают цифры 0-9 и буквы a, b, c, d, e, f. Шестнадцатеричные числа записываются с символами 0x перед числом, например, 0x25, 0xa1, 0xff. Запись вещественного литерала отличается от записи вещественного числа в математике тем, что вместо запятой, отделяющей целую часть от дробной, указывается точка, например, 123.34, -22.56. Кроме того, для записи вещественных чисел можно использовать так называемую экспоненциальную форму.

Кроме целых и вещественных значений в языке JavaScript могут встречаться так называемые логические значения. Существуют только два логических значения: истина и ложь. Первое представляется литералом true,

второе - false. В некоторых реализациях JavaScript может быть использована единица в качестве true, и ноль в качестве false.

Строковый литерал представляется последовательностью символов, заключенной в одинарные или двойные кавычки. Примером строкового литерала может быть строка "результат" или 'результат'.

Опр.: Элемент, используемый для хранения данных, называется переменной.

Тип переменной зависит от хранимых в ней данных, при изменении типа данных меняется тип переменной. Определить переменную можно с помощью оператора var, например: var test1.

В данном случае тип переменной test1 не определен и станет известен только после присвоения переменной некоторого значения. Оператор var можно использовать и для инициализации переменной, например, конструкцией var test2=276 определяется переменная test2 и ей присваивается значение 276.

Значение переменной изменяется в результате выполнения оператора присваивания. Оператор присваивания может быть использован в любом месте программы и способен изменить не только значение, но и тип переменной. Оператор присваивания выглядит так: a=b, где a - переменная, которой мы хотим задать некоторое значение; b - выражение, определяющее новое значение переменной.

Пусть в сценарии описаны следующие переменные:

```
let n=3725
let x=2.75
let p=true
let s="Выполнение завершено"
```

Переменные n и x имеют тип number, тип переменной p - логический, переменная s имеет тип string. В JavaScript определен тип function для всех стандартных функций и функций, определяемых пользователем. Объекты JavaScript имеют тип данных object. Переменные типа object часто называют просто объектами, они могут хранить объекты.

Выражения строятся из литералов, переменных, знаков операций, скобок. В зависимости от типа вычисленного значения выражения можно разделить на арифметические, логические и строковые.

Операции отношения применимы к операндам любого типа. Результат операции - логическое значение true, если сравнение верно, и false - в противном случае.

Приоритет операций определяет порядок, в котором выполняются операции в выражении. Сценарии, написанные на языке JavaScript, могут располагаться непосредственно в HTML-документе между тегами <script> и </script>.

Итак, для размещения сценария в HTML-документе следует написать следующее:

```
<script></script>
```

Документ может содержать несколько тегов `<script>`. Все они последовательно обрабатываются интерпретатором JavaScript. В следующем примере в раздел `<body>` (в тело) HTML-документа вставлены операторы языка JavaScript.

Пример 1. Вычисление площади треугольника

Необходимо написать сценарий, определяющий площадь прямоугольного треугольника по заданным катетам. Сценарий разместим в разделе `<body>` HTML-документа (листинг 1).

Листинг 1. Первый сценарий в документе :

```
<!DOCTYPE html>
<html>

<head>
  <title>Первый сценарий в документе</title>
</head>

<body>
  <p>
    Страница, содержащая сценарий.
  </p>
  <script>
    /*Инициализируются две переменные*/
    const a = 8;
    const h = 10;
    /*Для формирования вывода используется метод write объекта
document*/
    document.write("Площадь прямоугольного треугольника равна ", a * h
/ 2, ".");
  </script>
  <p>
    Конец формирования страницы, содержащей сценарий.
  </p>
</body>

</html>
```


Результат в браузере:

Страница, содержащая сценарий.

Площадь прямоугольного треугольника равна 40.

Конец формирования страницы, содержащей сценарий

Задания

1. Проверить пример из лабораторной работы.
2. Составить сценарий, в котором вычисляется площадь круга по заданному радиусу.
3. Составить сценарий, вычисляющий гипотенузу по заданным катетам.
4. Изменить любой скрипт добавив в него диалоговые окна для ввода исходных данных (функция `prompt`). Использовать функцию `alert` для вывода результата.

Лабораторная работа №3

Функции и обработка события на JavaScript

Основным элементом языка JavaScript является функция. Описание функции имеет вид

```
function F (V) {S},
```

где F - идентификатор функции, задающий имя, по которому можно обращаться к функции; V - список параметров функции, разделяемых запятыми; S - тело функции, в нем задаются действия, которые нужно выполнить, чтобы получить результат. Необязательный оператор return определяет возвращаемое функцией значение. Обычно все определения и функции задаются в разделе <head> документа. Это обеспечивает интерпретацию и сохранение в памяти всех функций при загрузке документа в браузер.

Пример 1. Нахождение площади треугольника.

В предыдущих примерах пользователю не предоставлялась возможность вводить значения, и в зависимости от них получать результат. Интерактивные документы можно создавать, используя формы. Предположим, что мы хотим создать форму, в которой поля Основание и Высота служат для ввода соответствующих значений. Кроме того, в форме создадим кнопку Вычислить. При щелчке мышью по этой кнопке мы хотим получить значение площади треугольника. Действие пользователя (например, щелчок кнопкой мыши) вызывает событие. События в основном связаны с действиями, производимыми пользователем с элементами форм HTML. Перехват и обработку событий можно задавать в параметрах элементов форм. Имя параметра обработки события начинается с приставки on, за которой следует имя самого события. Например, параметр обработки события click будет выглядеть как onclick.

Листинг 1. Реакция на событие Click.

```
<!DOCTYPE html>
<html lang="en">
  <html>
    <head>
      <title>Обработка значений из формы</title>
      <style>
        * {
          font-family: Verdana, Geneva, Tahoma, sans-serif;
          font-size: 16px;
        }
      </style>
    </head>
  </html>
</html>
```

```

p {
  color: rgb(0, 110, 255);
  font-size: 18px;
}

#calc-button {
  margin: 10px 0px;
  padding: 8px;
  border: none;
  border-radius: 5px;
  background-color: rgb(168, 205, 255);
}

#calc-button:hover {
  cursor: pointer;
  background-color: rgb(0, 110, 255);
  color: white;
}
</style>
</head>

<body>
  <p>Пример сценария со значениями из формы</p>
  <form>
    <label> Основание: </label>
    <input type="text" size="5" id="base" />
    <br /><br />
    <label> Высота: </label>
    <input type="text" size="5" id="height" />
    <br /><br />
    <input id="calc-button" type="button" value="Вычислить"
onClick="calculateTriangleArea()" />
  </form>

  <script>
    const baseInput = document.querySelector('#base');
    const heightInput = document.querySelector('#height');

    function calculateTriangleArea() {
      const base = +baseInput.value;
      const height = +heightInput.value;
      const area = (base * height) / 2;

      alert(`Площадь прямоугольного треугольника равна ${area}`);
      return area;
    }
  </script>

```

```

    </body>
  </html>
</html>

```

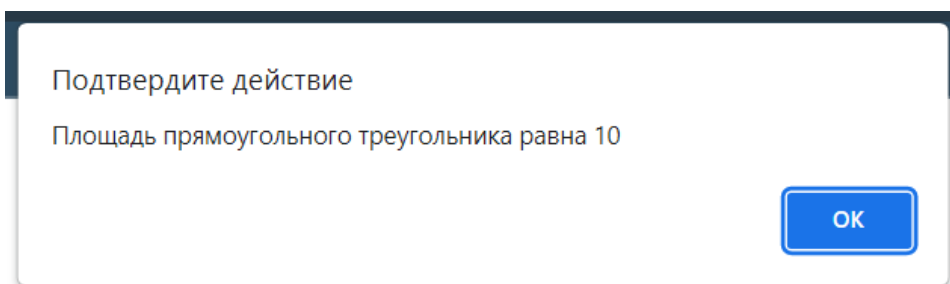
Результат в браузере:

Пример сценария со значениями из формы

Основание:

Высота:

Вычислить



При интерпретации HTML-страницы браузером создаются объекты JavaScript. Взаимосвязь объектов между собой представляет иерархическую структуру. На самом верхнем уровне иерархии находится объект `windows`, представляющий окно браузера. Объект `windows` является предком или родителем" всех остальных объектов. Каждая страница кроме объекта `windows` имеет объект `document`. Свойства объекта `document` определяются содержимым самого документа: цвет фона, цвет шрифта и т. д. Для получения значения основания треугольника, введенного в первом поле формы, должна быть выполнена конструкция:

```
const baseInput = document.querySelector('#base');
```

Метод `elem.querySelector(css)` возвращает первый элемент, соответствующий данному CSS-селектору.

Пример 2. Вычисление площади квадрата.

Напишем сценарий, определяющий площадь квадрата по заданной стороне. Площадь должна вычисляться в тот момент, когда изменилось

значение его стороны. Пусть форма содержит два текстовых поля: одно для длины стороны квадрата, другое для вычисленной площади. Кнопка Обновить очищает поля формы. Площадь квадрата вычисляется при возникновении события `change`, которое происходит в тот момент, когда значение элемента формы с именем `num1` изменилось, и элемент потерял фокус. HTML-код представлен в примере 2.

Листинг 2. Реакция на событие Change

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Обработка события Change - изменение значения
элемента</title>
    <style>
      * {
        font-family: Verdana, Geneva, Tahoma, sans-serif;
        font-size: 16px;
      }

      p {
        color: rgb(104, 32, 172);
        font-size: 18px;
      }

      #reset-button {
        margin: 10px 0px;
        padding: 8px;
        border: none;
        border-radius: 5px;
        background-color: rgb(228, 199, 255);
      }

      #reset-button:hover {
        cursor: pointer;
        background-color: rgb(104, 32, 172);
        color: white;
      }
    </style>
  </head>

  <body>
    <p>Вычисление площади квадрата</p>
    <form>
      Сторона: <input type="text" size="7" id="side"
onChange="calcSquareArea()" /> <br /><br />
      Площадь: <input type="text" size="7" id="result" /> <br /><br />
```

```
<input id="reset-button" type="reset" value="Обновить" />
</form>

<script>
  const sideInput = document.querySelector('#side');
  const resultField = document.querySelector('#result');

  function calcSquareArea() {
    const sideLength = sideInput.value;
    resultField.value = sideLength ** 2;
  }
</script>
</body>
</html>
```

Результат в браузере:

Вычисление площади квадрата

Сторона:

Площадь:

Событие Focus возникает в момент, когда пользователь переходит к элементу формы с помощью клавиши <Tab> или щелчка мыши. Событие "потеря фокуса" (Blur) происходит в тот момент, когда элемент формы теряет фокус. Событие select вызывается выбором части или всего текста в текстовом поле. Например, щелкнув дважды мышью по полю, мы выделим поле, наступит событие select, обработка которого приведет к вычислению требуемого значения. В языке JavaScript определены некоторые стандартные объекты и функции, пользоваться которыми можно без предварительного описания. Одним из стандартных объектов является объект Math. В свойствах упомянутого объекта хранятся основные математические константы, а его методы можно использовать для вызова основных математических функций. Выражение $y = \log x$ запишется $y = \text{Math.log}(x)$.

Задания

1. Проверить примеры из лабораторной работы.
2. На плоскости заданы координаты трех точек. Напишите сценарий, который вычисляет площадь треугольника (использовать событие Focus).
3. Напишите сценарий, который для точки, заданной координатами на плоскости, определяет расстояние до начала координат (использовать событие Select).
4. Напишите сценарий, который обменивает местами значения двух введенных переменных (использовать событие Blur).

Лабораторная работа №4

Операторы ветвлений и циклов, логические операции в JavaScript

При составлении программы часто необходимо выполнение различных действий в зависимости от результатов проверки некоторых условий. Для организации ветвлений можно воспользоваться условным оператором, который имеет вид:

```
if B {S1} else {S2}
```

где *B* - выражение логического типа; *S1* и *S2* - операторы. Выполнение условного оператора осуществляется следующим образом. Вычисляется значение выражения *B*. Если оно истинно, то выполняются операторы *S1*, если ложно - операторы *S2*. Если последовательность операторов *S1* или *S2* состоит лишь из одного оператора, то фигурные скобки можно опустить. Возможна сокращенная форма условного оператора:

```
if B {S}
```

где *B* - выражение логического типа; *S* - последовательность операторов. Выполнение краткого условного оператора осуществляется так: вычисляется значение выражения *B*, если оно истинно, то выполняются операторы *S*.

Пример 1. Нахождение максимального значения

Для трех заданных значений *a*, *b*, *c* необходимо написать сценарий, определяющий максимальное значение. Поступим следующим образом. Сначала максимальным значением *m* будем считать значение *a*, далее значение *b* сравним с максимальным. Если окажется, что *b* больше *m*, то максимальным становится *b*. И, наконец, значение *c* сравнивается с максимальным значением из предыдущих значений *a* и *b*. Если *c* больше *m*, то максимальным становится *c*. Выполнение инструкции

```
resultField.value = m;
```

обеспечивает запись вычисленного максимального значения в соответствующее поле формы. Функция `Number(s)` преобразует объект *s* в число. Полностью сценарий может быть записан так, как представлено в листинге 1.

Листинг 1. Вычисление максимального значения из трех заданных

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
    <title>Document</title>
```



```
<style>
  * {
    font-family: 'Comic Sans MS', 'Comic Sans', cursive;
  }

  form * {
    font-size: 18px;
  }

  h2 {
    text-align: center;
  }

  form {
    background: linear-gradient(45deg, rgba(211, 255, 167, 0.5),
    rgba(255, 236, 125, 0.5), rgba(255, 255, 255, 0.5), rgba(255, 236,
    125, 0.5), rgba(211, 255, 167, 0.5));
    width: 50%;
    margin: 20px auto;
    padding: 20px;
    border-radius: 18px;
  }

  input {
    margin: 8px 5px;
  }

  .button {
    border: 1px solid rgb(151, 151, 151);
    border-radius: 5px;
    padding: 8px;
    background-color: rgba(255, 255, 255, 0.5);
  }

  .button:hover {
    background-color: white;
    cursor: pointer;
  }
</style>
</head>

<body>
  <h2>Вычисление максимального значения</h2>
  <form>
    <label>Число 1:</label>
    <input type="number" id="num1" />
    <br />
    <label>Число 2:</label>
```

```
<input type="number" id="num2" />
<br />
<label>Число 3:</label>
<input type="number" id="num3" />
<br />
<br />
<label>Максимальное значение равно </label>
<br />
<input type="number" size="14" id="result" />
<input class="button" type="button" value="Определить"
onClick="getMaxValue()" />
<br />
<input class="button" type="reset" />
</form>

<script>
const num1 = document.querySelector('#num1');
const num2 = document.querySelector('#num2');
const num3 = document.querySelector('#num3');
const resultField = document.querySelector('#result');

function getMaxValue() {
  const a = Number(num1.value);
  const b = Number(num2.value);
  const c = Number(num3.value);
  let m = a;
  if (b > m) {
    m = b;
  }
  if (c > m) {
    m = c;
  }

  resultField.value = m;
}
</script>
</body>
</html>
```

Результат в браузере:

Вычисление максимального значения

Число 1:

Число 2:

Число 3:

Максимальное значение равно

Решим рассмотренную задачу другим способом. Вспомним, что стандартный объект `Math` имеет метод `max`, который определяет наибольшее значение двух аргументов. Опишем функцию `maxval1`, которая определяет максимальное значение из трех заданных значений и использует объект `Math`.

```
function getMaxValue() {  
    const a = Number(num1.value);  
    const b = Number(num2.value);  
    const c = Number(num3.value);  
    resultField.value = Math.max(Math.max(a, b), c);  
}
```

Если бы требовалось определить максимальное из четырех заданных значений `a`, `b`, `c`, `d`, то можно было бы воспользоваться формулой `Math.max(Math.max(a,b), Math.max(c,d))`.

Задания

1. Проверьте пример из лабораторной работы.
2. Вводится последовательность из пяти чисел. Напишите сценарий, в котором определяется число максимальных элементов.
3. Напишите программу, которая определяет, можно ли построить треугольник с заданными длинами сторон.
4. Точка на плоскости задается своими координатами. Определите, какой из четвертей прямоугольной системы координат принадлежит заданная точка.

Для успешного решения широкого круга задач требуется многократно повторить некоторую последовательность действий, записанную в программе один раз. В том случае, когда число повторений последовательности действий нам неизвестно, либо число повторений зависит от некоторых условий, можно воспользоваться оператором цикла вида:

```
while (B) {s}
```

где B - выражение логического типа; s - операторы, называемые телом цикла. Операторы s в фигурных скобках выполняются до тех пор, пока условие B не станет ложным.

Пример 1. Нахождение общего делителя

Напишем программу, которая для двух заданных чисел определяет наибольший общий делитель.

При решении задачи воспользуемся алгоритмом Евклида. Если значение m равно нулю, то наибольший общий делитель чисел n и m равен n:

$$\text{НОД}(n, 0) = n.$$

В остальных случаях верно следующее соотношение:

$$\text{НОД}(n, m) = \text{НОД}(m, n \% m).$$

В функции pod переменная r используется для получения остатка от деления чисел n и m (листинг 1). Выполнение цикла продолжается до тех пор, пока значение r не станет равным нулю. Последнее вычисленное значение m равно наибольшему общему делителю.

Листинг 1. Наибольший общий делитель двух чисел

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
```

```
<meta name="viewport" content="width=device-width, initial-
scale=1.0">
<title>Наибольший общий делитель двух чисел</title>
</head>
<style>
* {
  font-family: Verdana, Geneva, Tahoma, sans-serif;
  font-size: 17px;
}

input {
  margin: 8px 5px;
}

.button {
  padding: 8px 15px;
  border: none;
  border-radius: 5px;
  background-color: rgb(161, 226, 133);
  cursor: pointer;
}
</style>

<body>
<h4>Наибольший общий делитель двух заданных чисел</h4>

<form name="form1">
  <label>
    Введите число
  </label>
  <input type="text" class="num1" size="8">
  <br>

  <label>
    Введите число
  </label>
  <input type="text" class="num2" size="8">
  <br>

  <label>
    Наибольший общий делитель
  </label>
  <input type="text" class="res" size="8">
```

```
<input class="button" type="button" value="Вычислить"
onClick="nod()">
  <br>
  <hr>
  <input class="button" type="reset" value="Отменить">
</form>
<script>
function nod() {
  const num1 = document.querySelector('.num1');
  const num2 = document.querySelector('.num2');

  let n = num1.value;
  let m = num2.value;
  let p = n % m;
  while (p != 0) {
    n = m;
    m = p;
    p = n % m;
  }

  const result = document.querySelector('.res');
  result.value = m;
}
</script>
</body>

</html>
```

Результата в браузере:

Наибольший общий делитель двух заданных чисел

Введите число

Введите число

Наибольший общий делитель

Вычислить

Отменить

Если число повторений заранее известно, то можно воспользоваться следующим оператором цикла, который часто называют оператором цикла арифметического типа. Синтаксис этого оператора таков:

```
for (A; B; I){S}
```

Выражение *A* служит для инициализации параметра цикла и вычисляется один раз в начале выполнения цикла. Выражение *B* (условие продолжения) управляет работой цикла. Если значение выражения ложно, то выполнение цикла завершается, если истинно, то выполняется оператор *S*, составляющий тело цикла. Выражение *I* служит для изменения значения параметра цикла. После выполнения тела цикла *S* вычисляется значение выражения *I*, затем опять вычисляется значение выражения *B* и т.д. Цикл может прекратить свою работу в результате выполнения оператора `break` в теле цикла.

Пример 2. Совершенные числа

Напишем программу, определяющую, является ли заданное число *n* совершенным.

Совершенным называется число *n*, равное сумме своих делителей, не считая самого числа. Например, число 6 является совершенным, т. к. верно $6 = 1 + 2 + 3$, где 1, 2, 3 - делители числа 6. Число 28 также является совершенным, справедливо равенство $28 = 1 + 2 + 4 + 7 + 14$. При решении задачи воспользуемся только функцией `sumdei` (листинг 2).

Листинг 2. Итерационные методы. Совершенные числа

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
  <title>Document</title>
  <style>
    * {
      font-size: 17px;
      font-family: 'Gill Sans', 'Gill Sans MT', Calibri, 'Trebuchet
MS', sans-serif;
    }
  </style>
</head>
<body>
  <div style="text-align: center;">
    <h1>Совершенные числа</h1>
  </div>
</body>
</html>
```

```

.button {
  border: none;
  padding: 7px;
  margin: 0 5px;
  border-radius: 8px;
  background: linear-gradient(45deg, rgb(255, 241, 135), rgb(0,
227, 234));
}
</style>
</head>

<body>
  <p>Итерационные методы. Совершенные числа</p>
  <form>
    <label>Введите натуральное число:</label>
    <input type="text" size="8" id="number" />
    <input type="button" value="Выполнить" onClick="showNumberkind()"
class="button" />
    <hr />
    Данное число: <input type="text" size="24" id="result" readonly />
    <hr />
    <input type="reset" value="Отменить" class="button" />
  </form>

  <script language="JavaScript">
    function findSumOfDivisors(number) {
      let sum = 1;

      for (var i = 2; i <= number / 2; i++) {
        if (number % i == 0) {
          sum += i;
        }
      }

      return sum;
    }

    function isNumberPerfect(number) {
      return number == findSumOfDivisors(number);
    }
    const numberField = document.querySelector('#number');
    const resultField = document.querySelector('#result');

    function showNumberkind() {
      const number = +numberField.value;

```



```
    const numberKind = isNumberPerfect(number) ? 'совершенное' : 'не  
является совершенным';  
    resultField.value = numberKind;  
  }  
</script>  
</body>  
  
</html>
```

Результат в браузере:

Итерационные методы. Совершенные числа

Введите натуральное число:

Данное число:

Пример 3. Определение свойств элемента формы

Напишем сценарий, с помощью которого можно определить свойства элемента формы "поле ввода многострочного текста".

Свойства объекта с помощью оператора цикла формируются в строке `result`, затем после просмотра всех свойств значение строки `result` помещается в поле ввода многострочного текста.

Сценарий определения свойств текстового поля приведен в листинге 3.

Листинг 3. Операции над объектами. Свойства текстового поля

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Операции над объектами. Свойства текстового поля</title>
</head>

<body>
  <h4>
    Определение свойств объектов
  </h4>
  <form id="form1">
    <input type="button" value=Выполнить id="run-button">
    <hr>
    <textarea name="data" cols=30 rows=10 id=1>
      Текст
```

```
</textarea>
<hr>
<input type="reset" value=Очистить>
<script>
  const form = document.querySelector('#form1');
  const runButton = document.querySelector('#run-button');
  runButton.addEventListener('click', () => propobj(form));

  function propobj(obj) {
    var result = ""
    for (var i in obj) { result += obj.data.value + "." + i + " =
" + (obj.data)[i] + "\r\n" }
    result += "\n\r"
    form1.data.value = result
  }
</script>

</body>
</html>
```

Задания

1. Проверить примеры из лабораторной работы.
2. Напишите программу, которая "переворачивает" заданное натуральное число.
3. Напишите сценарий, в котором определяется количество "счастливых" шестизначных автобусных билетов, т. е. таких, в номерах которых сумма первых трех цифр равна сумме трех последних.
4. Напишите программу, определяющую все делители заданного натурального числа.

Лабораторная работа №5

Методы в JavaScript

Во время интерпретации HTML-документа браузером создаются объекты JavaScript. Свойства объектов определяются параметрами тегов языка HTML. Структура документа отражается в иерархической структуре объектов, соответствующих HTML-тегам. Родителем всех объектов является объект `windows`, расположенный на самом верхнем уровне иерархии, он представляет окно браузера и создается при запуске браузера. Для того чтобы открыть новое окно в сценарии JavaScript и отобразить в нем новый документ, применяется метод `open`, для закрытия окна можно воспользоваться методом `close`. Метод `alert` объекта `windows` отображает диалоговое окно с текстом, переданным методу в качестве параметра. Данный метод используется в случаях проверки правильности вводимых данных с помощью формы. Свойства объекта `windows` относятся ко всему окну, в котором отображается документ.

Подчиненными объектами (или объектами нижнего уровня) являются объекты `document`, `history`, `location`, `frame`. Свойства объекта `history` представляют адреса ранее загружаемых HTML-страниц. Свойства объекта `location` связаны с URL-адресом отображаемого документа, объекта `frame` - со специальным способом представления данных.

Свойства объекта `document` определяются содержимым самого документа: шрифт, цвет фона, формы, изображения и т. д. Объект `document` в зависимости от своего содержимого может иметь объекты, являющиеся для него подчиненными или дочерними. В частности, подчиненными для объекта `document` являются объекты `form`, `image`, `link`, `area` и др.

Для каждой страницы создается один объект `document`, некоторые его свойства соответствуют параметрам тега `<BODY>`: `bgColor`, `fgcolor`, `linkcolor`, `alinkcolor`, `vlinkColor`. Методы `write` и `writeln` записывают в документ текст, задаваемый параметром.

Если документ содержит изображения, то доступ к объекту, определяющему изображение, можно получить с помощью переменной, указанной в параметре `name` тега ``. Объект `image` имеет свойство `images`, которое содержит ссылки на все изображения, расположенные в документе. Ссылки перенумерованы, начиная с нуля. Доступ к первому изображению можно получить с помощью составной конструкции `document.images[0]`, ко второму - `document.images[1]`. Если на странице пять изображений, то доступ к последнему изображению можно получить, воспользовавшись ссылкой `document.images[4]`.

Рассмотрим примеры, в которых используются различные свойства объектов.

Для встраивания изображений в HTML-документ служит тег ``, имеющий обязательный параметр `src`, определяющий URL-адрес файла с

изображением. Можно задавать размеры выводимого изображения. Значение параметра `width` определяет ширину изображения, значение параметра `height` - высоту изображения. Значения параметров ширины и высоты могут не совпадать с истинными размерами изображений, тогда при загрузке изображения автоматически выполняется перемасштабирование.

Изображение можно поместить в рамку. Для этого используется параметр `border`. Значением параметра должно быть число, определяющее толщину рамки в пикселях. По умолчанию рамка вокруг изображения отсутствует, если только изображение не является ссылкой.

Параметр `alt` определяет альтернативный текст. При наведении курсора мыши на изображение появляется комментарий.

Пример 1. Перестановка изображений

Напишем сценарий, который реализует обмен рисунков в документе. Пусть в документе расположено три изображения, пронумерованных от 1 до 3. В текстовых полях указываются номера рисунков, которые необходимо поменять местами. Требуется, чтобы после нажатия кнопки Обменять изображения переместились на нужные места.

Сначала проверим, правильно ли заданы номера изображений, если это не так, то выдадим сообщение. Обмен изображений выполнен с использованием деструктуризации массива (ссылка: [Деструктуризация массива](#)). Здесь мы создаём временный массив из двух переменных и немедленно деструктурируем его в порядке замены.

Приведем полностью документ со сценарием (листинг 1).

Листинг 1. Перестановка изображений с помощью сценария

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
    <title>Document</title>
    <style>
      * {
        font-family: Verdana, Geneva, Tahoma, sans-serif;
        font-size: 18px;
      }

      input[type="button"] {
```

```

        background-color: antiquewhite;
        border: 2px solid burlywood;
    }

    img {
        width: 100px;
        margin: 5px;
        border: 1px solid black;
        background-color: beige;
    }
</style>
</head>
<body>
    <h4>Галерея рисунков</h4>
    
    
    

    <form>
        <label>Рисунки с номерами:</label>
        <input type="text" id="first-image-number" size="1" />
        <input type="text" id="second-image-number" size="1" />
        <input type="button" value="Поменять местами"
onClick="swapImages()" />
    </form>

    <script>
        const firstImageNumberInput = document.querySelector('#first-
image-number');
        const secondImageNumberInput = document.querySelector('#second-
image-number');

        function swapImages() {
            const firstImageNumber = +firstImageNumberInput.value;
            const secondImageNumber = +secondImageNumberInput.value;
            const images = document.images;

            if (firstImageNumber < 1 || firstImageNumber > 3 ||
secondImageNumber < 1 || secondImageNumber > 3) {
                alert('Неверно заданы номера рисунков!');
            }

            const firstImage = images[firstImageNumber - 1];
            const secondImage = images[secondImageNumber - 1];

```

```

        [firstImage.src, secondImage.src] = [secondImage.src,
firstImage.src];
    }
    </script>
</body>
</html>

```

Результат в браузере:

Галерея рисунков



Рисунки с номерами:

Пример 2. Простое вертикальное меню

Напишем сценарий, реализующий вертикальное графическое меню с появляющейся стрелкой возле пункта, у которого находится курсор.

При наведении курсора на элемент `` вызывается функция `itemMouseoverHandler()`, а когда курсор покидает элемент, то происходит вызов функции `itemMouseoutHandler()`. В первой функции происходит обращение к элементу, на котором произошло событие `mouseover`, через ключевое слово «`this`», далее происходит получение первого дочернего элемента, которым является элемент `` и у данного элемента удаляется класс `hide`, который значению `css` свойства `visibility` элементов, принадлежащим к этому классу, *устанавливает значение* `hidden`. А в функции `itemMouseoutHandler()` происходит добавление класса `hide`.

Документ со сценарием, реализующий вертикальное графическое меню, представлен в листинге 2.

Листинг 2. Простое вертикальное меню

Содержимое файла `script.js`:

```
const navItems = document.querySelectorAll("nav>ul>li");
```

```

navItems.forEach((item) => {
  item.onmouseover = itemMouseoverHandler;
  item.onmouseout = itemMouseoutHandler;
});

function itemMouseoverHandler() {
  this.firstChild.classList.remove('hide');
}

function itemMouseoutHandler() {
  this.firstChild.classList.add('hide');
}

```

Содержимое файла index.html:

```

<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
  <title>Document</title>
  <link rel="stylesheet" href="./style.css" />
</head>

<body>
  <nav>
    <ul>
      <li>
        
        <a href="#">Home</a>
      </li>
      <li>
        
        <a href="#">About</a>
      </li>
      <li>
        
        <a href="#">Clients</a>
      </li>
      <li>
        
        <a href="#">Contact Us</a>
      </li>
    </ul>
  </nav>

```



```
        </li>
    </ul>
</nav>
<script src="./script.js"></script>
</body>

</html>
```

Содержимое файл style.css:

```
* {
  padding: 0;
  margin: 0;
  font-family: "Comic Sans MS", "Comic Sans", cursive;
}

body{
  display: flex;
  height: 100vh;
}

nav {
  background: white;
  box-shadow: 0 3px 10px -2px rgba(0, 0, 0, 0.1);
  border: 1px solid rgba(0, 0, 0, 0.3);
  margin: 50px;
  padding: 5px;
  padding-right: 30px;
}

nav ul {
  width: 140px;
  height: 100%;
  display: flex;
  justify-content: space-evenly;
  flex-direction: column;
  list-style: none;
}

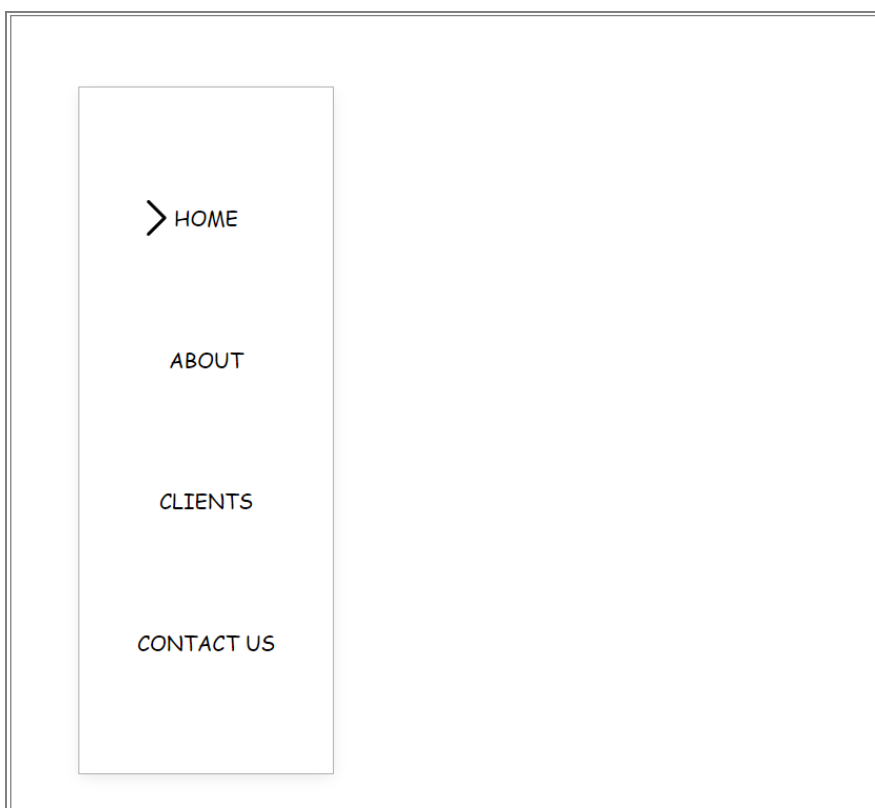
nav ul li {
  display: flex;
  justify-content: center;
  align-items: center;
}
```

```
nav ul li a {  
  display: block;  
  color: black;  
  font-size: 0.9em;  
  text-decoration: none;  
  text-transform: uppercase;  
}
```

```
img{  
  rotate: 90deg;  
  max-height: 25px;  
  max-width: 25px;  
}
```

```
.hide {  
  visibility: hidden;  
}
```

Результат в браузере:



Задания

1. Проверить примеры лабораторной работы.

2. Написать сценарий выбора из трех изображений одного, которое вставляется ниже этих трех.

3. Написать сценарий картинки с "эффектом приближения", т.е. увеличения размеров как реакция на попадание курсора мыши в поле рисунка (использовать свойства width и height).

4. Написать сценарий графического горизонтального меню с появляющейся стрелкой над пунктом, у которого находится курсор.

Лабораторная работа №6 Обработка массивов на JavaScript

Существует два варианта синтаксиса для создания пустого массива:

```
let arr = new Array();  
let arr = [];
```

Практически всегда используется второй вариант синтаксиса. В скобках мы можем указать начальные значения элементов:

```
let fruits = ["Яблоко", "Апельсин", "Слива"];
```

Общее число элементов массива содержится в его свойстве `length`:

```
alert( fruits.length ); // 3
```

Методы массивов:

Добавление/удаление элементов:

- `arr.push(...items)` – добавляет элементы в конец,
- `arr.pop()` – извлекает элемент из конца,
- `arr.shift()` – извлекает элемент из начала,
- `arr.unshift(...items)` – добавляет элементы в начало.

concat

Метод `arr.concat` создаёт новый массив, в который копирует данные из других массивов и дополнительные значения.

Его синтаксис:

```
arr.concat(arg1, arg2...)
```

Перебор: forEach

Метод `arr.forEach` позволяет запускать функцию для каждого элемента массива.

Его синтаксис:

```
arr.forEach(function(item, index, array) {  
  // ... делать что-то с item  
});
```

```
["Bilbo", "Gandalf", "Nazgul"].forEach((item, index, array) => {  
  alert(` ${item} имеет позицию ${index} в ${array} `);  
});
```

filter

Метод `filter()` создаёт новый массив со всеми элементами, прошедшими проверку, задаваемую в передаваемой функции..

```
let results = arr.filter(function(item, index, array) {
  // если true - элемент добавляется к результату, и перебор
  продолжается
  // возвращается пустой массив в случае, если ничего не найдено
});
```

Например:

```
let users = [
  {id: 1, name: "Вася"},
  {id: 2, name: "Петя"},
  {id: 3, name: "Маша"}
];

// возвращает массив, состоящий из двух первых пользователей
let someUsers = users.filter(item => item.id < 3);

alert(someUsers.length); // 2
```

map

Метод `arr.map` является одним из наиболее полезных и часто используемых.

Он вызывает функцию для каждого элемента массива и возвращает массив результатов выполнения этой функции.

Синтаксис:

```
let result = arr.map(function(item, index, array) {
  // возвращается новое значение вместо элемента
});
```

Например, здесь мы преобразуем каждый элемент в его длину:

```
let lengths = ["Bilbo", "Gandalf", "Nazgul"].map(item => item.length);
alert(lengths); // 5,7,6
```

reduce

Если нам нужно перебрать массив – мы можем использовать `forEach`, `for` или `for..of`.

Если нам нужно перебрать массив и вернуть данные для каждого элемента – мы используем `map`.

Методы `arr.reduce` похожи на методы выше, но они немного сложнее. Они используются для вычисления какого-нибудь единого значения на основе всего массива.

Синтаксис:

```
let value = arr.reduce(function(accumulator, item, index, array) {
  // ...
}, [initial]);
```

Функция применяется по очереди ко всем элементам массива и «переносит» свой результат на следующий вызов.

Аргументы:

- accumulator – результат предыдущего вызова этой функции, равен initial при первом вызове (если передан initial),
- item – очередной элемент массива,
- index – его индекс,
- array – сам массив.

При вызове функции результат её вызова на предыдущем элементе массива передаётся как первый аргумент.

Звучит сложновато, но всё становится проще, если думать о первом аргументе как «аккумулирующем» результат предыдущих вызовов функции. По окончании он становится результатом reduce.

Этот метод проще всего понять на примере.

Тут мы получим сумму всех элементов массива всего одной строкой:

```
let arr = [1, 2, 3, 4, 5];  
  
let result = arr.reduce((sum, current) => sum + current, 0);  
  
alert(result); // 15
```

Задания

1. Проверить пример 2 из лабораторной работы.
2. Создать простейший мультипликационный сюжет с использованием массива.
3. Задан одномерный массив вещественных чисел. Напишите сценарий, который определяет число положительных элементов массива.
4. Задан одномерный массив вещественных чисел. Напишите сценарий, позволяющий найти максимальный элемент в массиве.

Лабораторная работа №7

Использование элементов управления на JavaScript

Данные удобно представлять с помощью элемента управления "переключатель" (или "радиокнопка") в том случае, когда из нескольких вариантов может быть выбран лишь один.

Пример 1. Вычисление площади фигуры.

Необходимо выбрать форму фигуры и определить ее площадь. Пусть для выбора фигуры задана следующая форма:

```
<form id="form">
  <label>Введите значение</label>
  <input type="number" id="data" size="10" value="0"/>
  <hr>
  <label style="display: block; margin-bottom: 10px;">Укажите
форму</label>
  <input type="radio" name="figure" value="1"/>
  <label>Квадрат</label>
  <br>
  <input type="radio" name="figure" value="3.14"/>
  <label>Круг</label>
  <br>
  <input type="radio" name="figure" value="0.42"/>
  <label>Треугольник</label>
  <hr>
  <input type="reset" value="Отменить" />
  <hr>
  <label>Площадь:</label>
  <input type="text" id="result" size="10" />
</form>
```

В этой форме шесть элементов. Первый элемент служит для ввода строки текста. Следующие три элемента образуют группу и являются переключателями. Пятый элемент создает кнопку сброса, нажатие которой отменяет все сделанные изменения. Шестой элемент является элементом для ввода строки.

Так как объект forms имеет свойство-массив elements, в котором содержатся ссылки на элементы формы в порядке их перечисления в теге <FORM>, то получить доступ к первому элементу формы можно либо с помощью значения параметра name этого элемента (document.form1.data), либо используя объектную модель JavaScript (document.forms[0].elements[0]).

Второй элемент рассматриваемой формы можно получить, если воспользоваться конструкцией `document.forms[0].elements[1]`. Это элемент-переключатель, определенный в составе группы элементов. В рассматриваемом примере группа элементов состоит из трех переключателей. В одну группу входят элементы с одинаковым значением параметра `name`. Доступ к следующим элементам группы может быть осуществлен так: `document.forms[0].elements[2]`, `document.forms[0].elements [3]`. Обязательный параметр `value` должен иметь уникальное значение для каждого элемента группы. Пользователь может выбрать только один вариант.

Напишем сценарий, в котором в зависимости от длины стороны или радиуса и формы выбранной фигуры вычисляется ее площадь. Для простоты будем считать, что фигура может иметь либо форму квадрата (задается его сторона), либо форму круга (задается радиус), либо форму равностороннего треугольника (задается его сторона).

Площадь рассматриваемых фигур считается по формуле ka^2 , где k - коэффициент, зависящий от формы выбранной фигуры; a - задаваемое пользователем значение. Вычисления будут проще, если коэффициент k указать в качестве значения параметра `value` соответствующего переключателя. Щелчок на элементе "переключатель" соответствует событию `click`, обработка которого заключается в вызове функции `test`. Функция имеет единственный параметр, значение параметра - `value` переключателя, которое служит для вычисления площади фигуры.

HTML-код приведен в листинге 1.

Листинг 1. Вычисление площади выбранной с помощью переключателя фигуры

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
  <title>Document</title>
</head>
<style>
  * {
    font-family: Verdana, Geneva, Tahoma, sans-serif;
    font-size: 16px;
  }
```



```

input[type="reset"] {
  border: none;
  padding: 8px;
  border-radius: 5px;
  background-color: gold;
}
</style>

<body>
  <form id="form">
    <label>Введите значение</label>
    <input type="number" id="data" size="10" value="0"
onChange="displayFigureSquare()" />
    <hr>
    <label style="display: block; margin-bottom: 10px;">Укажите
форму</label>
    <input type="radio" name="figure" value="1"
onClick="displayFigureSquare()" />
    <label>Квадрат</label>
    <br>
    <input type="radio" name="figure" value="3.14"
onClick="displayFigureSquare()" />
    <label>Круг</label>
    <br>
    <input type="radio" name="figure" value="0.42"
onClick="displayFigureSquare()" />
    <label>Треугольник</label>
    <hr>
    <input type="reset" value="Отменить" />
    <hr>
    <label>Площадь:</label>
    <input type="text" id="result" size="10" />
  </form>

  <script>
    const valueInput = document.querySelector('#data');
    const resultField = document.querySelector('#result');
    const form = document.querySelector('#form');

    function getFigureSquare() {
      const k = +form.elements["figure"].value;
      const a = +valueInput.value;

      const square = k * Math.pow(a, 2);
      return square;
    }
  </script>

```

```

    }

    function displayFigureSquare() {
        const square = getFigureSquare();
        resultField.value = square;
    }
</script>
</body>

</html>

```

Результат в браузере:

Введите значение

Укажите форму

Квадрат
 Круг
 Треугольник

Площадь:

Пример 2. Выбор параметров обтекания изображения текстом

Напишем сценарий, который предоставляет возможность пользователю задавать значения параметров, определяющих, к какому полю окна (левому или правому) прижимается изображение, и, соответственно, с какой стороны текст его обтекает.

Если значение свойства `float` равно `Left`, то изображение прижимается к левому краю окна просмотра браузера, а текст или другие элементы документа "обтекают" изображение с правой стороны. Текст, размещаемый рядом с изображением, может занимать несколько строк. По умолчанию значение параметра `float` равно `Left`. При нажатии на кнопку Обновить для изображения и текста будут установлены значения параметров, принимаемых по умолчанию.

Пример HTML-кода, который содержит сценарий, обеспечивающий выполнение действий, задаваемых пользователем, приведен в листинге 2.

Листинг 2. Обтекание текстом изображения

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
  <title>Document</title>
  <link rel="stylesheet" href="style.css" />
</head>
<style>
  .image {
    height: 110px;
    margin: 5px 10px;
    float: left;
    background-color: azure;
  }

  .result-window {
    margin: 30px;
    padding: 15px;
    width: 350px;
    background-color: antiquewhite;
    border-radius: 10px;
    text-align: justify;
  }
</style>

<body>
  <h4>Изображение и текст. Обтекание изображения текстом</h4>

  <form name="form1">
    <p>Выберите значение параметра выравнивания и нажмите кнопку
"Просмотр".</p>

    <input type="radio" id="left" name="align" checked value="left" />

    <label for="left">(left) Изображение выравнивается по левому
краю</label>
    <br>

    <input type="radio" id="right" name="align" value="right" />
    <label for="right">(right) Изображение выравнивается по правому
краю</label>
```

```

    <br>
    <br>
    <input type="button" value="Просмотр"
onclick="changeBlockAligment()" />
    <input type="reset" value="Отменить" onclick="reset()" />
</form>

<div class="result-window">
    Иван Иванович Шишкин является одним из основоположников русского
национального пейзажа.
    </img>
    В полотне "Рожь" Шишкин создал образ большой эпической силы и
подлинно монументального звучания.
    Могучая, полная богатырских сил природа, богатый, привольный край.
    (Т. Юрова)
</div>

<script>
    const image = document.querySelector('.image');
    function changeBlockAligment() {
        const selectedValue =
document.querySelector('input[name="align"]:checked').value;
        image.style.float = selectedValue;
    }

    function reset() {
        image.style.float = 'left';
    }
</script>
</body>

</html>

```

Результат в браузере:

Изображение и текст. Обтекание изображения текстом

Выберите значение параметра выравнивания и нажмите кнопку "Просмотр".

- (left) Изображение выравнивается по левому краю
 (right) Изображение выравнивается по правому краю

Просмотр

Отменить

Иван Иванович Шишкин является одним из основоположников русского национального пейзажа. В полотне "Рожь" Шишкин создал образ большой эпической силы и подлинно монументального звучания. Могучая, полная богатырских сил природа, богатый, привольный край. (Т. Юрова)



полная богатырских сил природа, богатый, привольный край. (Т. Юрова)

Если изображение рассматривается как элемент строки, то значения параметров выравнивания задают расположение изображения относительно строки текста. Верхняя граница изображения может быть выровнена либо по самому высокому текстовому элементу текущей строки, либо по самому высокому элементу в строке (например, другому изображению). Базовой считается нижняя часть линии текста, которая проводится без учета нижней части некоторых символов. Середину изображения можно выровнять либо по базовой линии, либо по середине текущей строки. Нижнюю часть изображения можно выровнять по базовой линии, либо по нижней границе текущей строки.

Элемент управления "флажок" используется в случае, когда из предложенных вариантов можно выбрать как один, так и несколько. Каждый вариант выбора задается флажком, который можно либо установить, либо сбросить. Флажок определяется в теге `<input>` значением `checkbox` параметра `type`. Обязательным параметром является параметр `value`, значение которого будет передано на обработку в случае выбора нажатием кнопки.

Пример 3. Выбор характеристик издания

Предположим, читателю предлагается заполнить анкету, в которой требуется указать название любимого издания и выбрать из предложенного списка характеристики, которые присущи рассматриваемому изданию.

Для задания характеристик издания можно воспользоваться флажком. Пользователь устанавливает флажки для тех свойств, которыми, по его мнению, обладает издание. Обработка анкеты будет состоять в том, что выбранные свойства будут отражены в поле ввода многострочного текста.

При щелчке мышью по флажку возникает событие `click`, обработка которого состоит в вызове функции `set`. Для формирования строки результата служит переменная `str`; к имеющемуся значению добавляется значение параметра функции и помещается в текстовое поле. Если нажать ни кнопку Отмена, то очистятся все поля формы. Однако следует позаботиться о том, чтобы значение переменной `s` изменилось на начальное. Значение параметра реакции на событие `click` при щелчке по кнопке Отмена задается оператором присваивания, обеспечивающим начальные условия.

HTML-код представлен в листинге 1.

Листинг 3. Анкета читателя

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Анкета читателя</title>
</head>
<style>
  * {
    font-family: Cambria, Cochin, Georgia, Times, 'Times New Roman',
serif;
  }

  h3 {
    text-align: center;
  }

  h4 {
    font-style: italic;
  }

  div {
    font-size: 17px;
    border-radius: 5px;
  }
```

```
div input {
  margin: 8px 0;
}

div label {
  color: rgb(117, 0, 167);
  margin-left: 8px;
}

form {
  box-sizing: border-box;
  margin: 10px;
  padding: 0 20px 15px;
}

.second-form {
  border: 4px dashed gainsboro;
  padding: 0 20px 20px;
}

.first-form input {
  width: 100%;
  box-sizing: border-box;
}

textarea {
  margin: 30px 0 15px;
  font-size: 17px;
}

.button {
  font-size: 16px;
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  background-color: rgb(235, 189, 255);
  cursor: pointer;
}

section {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
}
```

```

    }
</style>

<body>
  <h3>Анкета читателя</h3>
  <section>
    <form class="first-form">
      <h4>
        Введите название любимого журнала или газеты
      </h4>
      <input type="text"><br>
    </form>

    <form class="second-form">
      <h4>Что Вас привлекает в издании?</h4>
      <div>
        <input type="checkbox" value="Стиль подачи материала">
        <label>
          Стиль подачи материала
        </label>
        <br>
        <input type="checkbox" value="Достоверность информации">
        <label>
          Достоверность информации
        </label>
        <br>
        <input type="checkbox" value="Дизайн и оформление">
        <label>
          Дизайн и оформление
        </label>
        <br>
      </div>
      <textarea cols=50 rows=7> </textarea>
      <br>
      <input class='button' type="reset" value="Отмена">
    </form>
  </section>

  <script>
    const checkboxes = document.querySelectorAll('.second-form
input[type="checkbox"]');

    checkboxes.forEach((checkbox) => {
checkbox.addEventListener('click', set) });

```



```

const firstLine = "Вас привлекает:\n";
let str = firstLine;

function set() {
  if (this.checked) {
    str = str + this.value + "\n";
    document.querySelector('.second-form textarea').value = str;
  }
}
</script>
</body>

</html>

```

Результат в браузере:

Анкета читателя

Введите название любимого журнала или газеты

Что Вас привлекает в издании?

- Стиль подачи материала
- Достоверность информации
- Дизайн и оформление

Вас привлекает:
Стиль подачи материала
Достоверность информации

Пример 4. Использование флажков в анкете переводчика

В анкете требуется указать те языки, которыми владеет переводчик. Предположим, что за знание каждого языка назначается определенная сумма. Размер вознаграждения определяется после заполнения анкеты в зависимости от тех языков, которыми пользователь владеет. По результатам заполненной

переводчиком анкеты напишите сценарий определения размера вознаграждения.

Для задания сведений о том, владеет ли пользователь определенным языком, удобно применять флажок. При щелчке мышью по кнопке Вознаграждение выполняется функция `grant()`. Требуется проанализировать состояние флажков. Свойство `checked` возвращает логическое значение, представляющее текущее значение отдельного флажка (`true` или `false`).

HTML-код представлен в листинге 2.

Листинг 4. Данные, представленные флажком. Анкета переводчика

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Данные, представленные флажком. Анкета переводчика</title>
  <style>
    * {
      font-family: Cambria, Cochin, Georgia, Times, 'Times New Roman',
serif;
    }

    section {
      display: flex;
      flex-direction: column;
      align-items: center;
    }

    form :first-child {
      margin-bottom: 10px;
      display: block;
    }

    form {
      border: 4px dashed gainsboro;
      padding: 20px;
      font-size: 18px;
    }

    input[type="text"] {
```

```

    font-size: 18px;
    width: 100%;
    box-sizing: border-box;
}

input[type="checkbox"] {
    margin: 8px 0;
}

.button {
    font-size: 18px;
    margin: 20px 0;
    padding: 8px;
    border: none;
    border-radius: 5px;
    background-color: rgb(168, 179, 255);
    cursor: pointer;
}
</style>
</head>

<body>
  <section>
    <h3>Анкета для переводчиков</h3>
    <form name="form1">
      <label>
        Укажите те языки, которыми Вы владеете в совершенстве:
      </label>

      <input type="checkbox" name="lan" value=100>
      <label>русский</label>
      <br>

      <input type="checkbox" name="lan" value=200>
      <label>английский</label>
      <br>

      <input type="checkbox" name="lan" value=300>
      <label>французский</label>
      <br>

      <input class="button" type="button" value=Вознаграждение
onClick="grant()">
      <br>

```

```

<input type="text" class="result">
<br>

<input class="button" type="reset" value="Отменить">
</form>
</section>

<script language="JavaScript">

function grant() {
  const checkboxes =
document.querySelectorAll('input[type="checkbox"]');

  let award = 0;
  checkboxes.forEach((checkbox) => {
    if (checkbox.checked) {
      award = award + Number(checkbox.value)
    }
  });

  document.querySelector('.result').value = "Вам полагается
вознаграждение " + award + " у.е."
}

</script>

</body>
</html>

```

Результат в браузере:

Анкета для переводчиков

Укажите те языки, которыми Вы владеете в совершенстве:

русский

английский

французский

Вознаграждение

Вам полагается вознаграждение 500 у.е.

Отменить

Упражнение

Напишите сценарий обработки анкеты слушателя курсов. Пользователь может выбрать курс, его продолжительность, язык, на котором он готов работать с преподавателем, и форму отчетности. В зависимости от этих параметров определяется стоимость отдельного курса и стоимость всего обучения. Анкета приведена на рис. 2.

ФИО *данное поле обязательно для заполнения
 e-mail
 Возраст
 Адрес

Название курсов	Продолжительность	Язык	Отчетность	Стоимость
<input checked="" type="checkbox"/> Информатика	<input type="radio"/> 36 <input checked="" type="radio"/> 64 <input type="radio"/> 128	<input checked="" type="radio"/> Русский <input type="radio"/> Английский	<input type="radio"/> Экзамен <input checked="" type="radio"/> Зачет	<input type="text" value="Стоимость 18 у.е."/>
<input checked="" type="checkbox"/> Базы данных	<input checked="" type="radio"/> 36 <input type="radio"/> 64 <input type="radio"/> 128	<input checked="" type="radio"/> Русский <input type="radio"/> Английский	<input checked="" type="radio"/> Экзамен <input type="radio"/> Зачет	<input type="text" value="Стоимость 6 у.е."/>
<input type="checkbox"/> Сети	<input checked="" type="radio"/> 36 <input type="radio"/> 64 <input type="radio"/> 128	<input checked="" type="radio"/> Русский <input type="radio"/> Английский	<input type="radio"/> Экзамен <input checked="" type="radio"/> Зачет	<input type="text" value="Стоимость 0 у.е."/>
<input type="checkbox"/> Логика	<input checked="" type="radio"/> 36 <input type="radio"/> 64 <input type="radio"/> 128	<input checked="" type="radio"/> Русский <input type="radio"/> Английский	<input type="radio"/> Экзамен <input checked="" type="radio"/> Зачет	<input type="text" value="Стоимость 0 у.е."/>
Общая стоимость 24 у.е.				

Лабораторная работа №8

Создание простой программы на PHP. Использование основных операторов PHP

Цель работы: Получение навыков работы с PHP. Изучение основных операторов PHP

Порядок выполнения работы.

Изучить теоретические сведения.

Выполнить задание к лабораторной работе в соответствии с вариантом.

Оформить отчет.

Требования к отчету.

Цель работы.

Постановка задачи.

Текст программы.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Что нужно, чтобы запустить PHP-скрипт?

Программирование на PHP отличается от программирования на других языках достаточно сложной системой настройки среды. Первым шагом в освоении Web- технологий является воспроизведение рабочей среды на компьютере разработчика. Для этого потребуется запустить и настроить несколько программных компонентов.

Доступность Web-приложений и HTML-файлов в сети Интернет обеспечивается Web-сервером, который по протоколу HTTP выдает их любому клиенту, правильно оформившему запрос. Наиболее популярным сервером, используемым совместно с PHP, долгие годы остается Web-сервер Apache.

Интерпретатор PHP представляет собой либо внешнюю CGI-программу, либо динамическую библиотеку, которую необходимо подключить к Web-серверу, чтобы вместо кода PHP-скриптов клиенту выдавались результаты его выполнения. Ситуация осложняется тем, что к PHP-интерпретатору могут подключаться различные расширения, также оформленные в виде динамических библиотек. Для большинства современных приложений требуется довольно много внешних расширений, поэтому без ручного редактирования конфигурационных файлов довольно трудно обойтись.

Практически ни одна крупная программа не обходится без использования базы данных. Традиционно совместно с Web-сервером Apache и интерпретатором PHP используется СУБД MySQL.

Так как и язык программирования PHP, и Web-сервер Apache, и MySQL-сервер первоначально разработаны для UNIX-подобных операционных систем, их настройка и администрирование сводятся к редактированию конфигурационных файлов и работе в командной строке. Такой подход часто сбивает с толку программистов, не имеющих опыта работы в UNIX.

6.2 Можно ли обойтись без утомительной настройки серверов и PHP

Конфигурирование серверов и PHP, настройка режима их работы являются важными разделами, позволяющими увереннее чувствовать себя Web-разработчикам, быстро осуществлять локализацию ошибок, попыток взлома, настраивать наиболее эффективную работу Web-приложений. Однако даже локальная настройка Web-серверов, а тем более настройка их для работы в сети Интернет, является отдельной областью деятельности, требующей зачастую не меньшей отдачи, чем программирование. Поэтому Web-разработчики зачастую отказываются от изучения конфигурирования серверов и их взаимодействия, оставляя эту область системным администраторам. Чтобы не завязнуть в многочисленных настройках серверов, правилах их связывания друг с другом, Web-разработчики прибегают к готовым пакетам, где вся настройка выполнена профессиональными администраторами. Такой пакет остается только загрузить и установить, после чего можно сразу приступить к работе с языком программирования. Наиболее популярными на сегодняшний день пакетами, объединяющими интерпретатор PHP, Web-сервер Apache и СУБД MySQL, являются Denwer, загрузить который можно по ссылке <http://www.denwer.ru/>, MAMP и OpenServer. Все эти пакеты находятся в свободном доступе в сети Интернет.

Традиционно тестирование любой программы PHP начинается с фразы "Hello world!".

Рассмотрим пример программы PHP.

```
echo "Hello world!";
```

Запустим сценарий. Легко убедиться, что он действительно работает, да к тому же еще и безотказно.

В окне веб-браузера это будет выглядеть [ТАК](#).

Если мы напишем

```
<body>
Hello world!
</body>
```

то получим тот же результат.

В окне веб-браузера это будет выглядеть [ТАК](#).

Думаете, что произошла ошибка и редактор вместо примера кода на РНР случайно вставил в текст пример HTML-страницы? Тут действительно нет вообще никаких операторов РНР, и содержимое файла с "программой" состоит целиком из статического текста. Выходит, что обычный HTML-текст также правильно обрабатывается РНР.

Рассмотрим чуть более сложный пример.

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Простой сценарий на РНР</title>
</head>

<body>
<h1>Здравствуйтесь!</h1>
<?php
// Вычисляем текущую дату в формате "день.месяц год"
$dat=date("d.m y");
// Вычисляем текущее время
$tm=date("h:i:s");
# Выводим их
echo "Текущая дата: $dat года<br>\n";
echo "Текущее время: $tm<br>\n";
# Выводим цифры
echo "А вот квадраты и кубы первых 5 натуральных
чисел:<br>\n";
for($i=1; $i<=5; $i++)
{ echo "<li>$i в квадрате = ".$i*$i);
echo ", $i в кубе = ".$i*$i*$i)."\n";
}
?>
</body>
</html>
```


В окне веб-браузера это будет выглядеть

Здравствуйтесь!

Текущая дата: 01.09 20 года

Текущее время: 08:35:39

А вот квадраты и кубы первых 5 натуральных чисел:

- 1 в квадрате = 1, 1 в кубе = 1
- 2 в квадрате = 4, 2 в кубе = 8
- 3 в квадрате = 9, 3 в кубе = 27
- 4 в квадрате = 16, 4 в кубе = 64
- 5 в квадрате = 25, 5 в кубе = 125

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Простой сценарий на PHP</title>
</head>

<body>
<h1>Здравствуйтесь!</h1>
Текущая дата: 01.09 20 года<br>
Текущее время: 08:35:39<br>
А вот квадраты и кубы первых 5 натуральных чисел:<br>
<li>1 в квадрате = 1, 1 в кубе = 1
<li>2 в квадрате = 4, 2 в кубе = 8
<li>3 в квадрате = 9, 3 в кубе = 27
<li>4 в квадрате = 16, 4 в кубе = 64
<li>5 в квадрате = 25, 5 в кубе = 125

</body>
</html>
```

Синтаксис любого языка программирования гораздо легче "почувствовать" на примерах, нежели используя какие-то диаграммы и схемы.

Приступим к разбору программы.

Начало сценария, если бы не был уже затронут второй пример, может озадачить: разве это сценарий? Откуда HTML-тэги `<html>` и `<body>`? Вот тут-то и кроется главная особенность (кстати, чрезвычайно удобная) языка

PHP: PHP-скрипт может вообще не отличаться от обычного HTML-документа, как мы это уже заметили ранее.

Идем дальше. Код сценария начинается после открывающего тэга `<?php` и заканчивается закрывающим `?>`. Итак, между этими двумя тэгами текст интерпретируется как программа, и в HTML-документ не попадает. Если же программе нужно что-то вывести, она должна воспользоваться оператором `echo` (это не функция, а конструкция языка: ведь, в конце концов, если это функция, то где же скобки?). Мы подробно рассмотрим ее работу в дальнейшем. Итак, PHP устроен так, что любой текст, который расположен вне программных блоков, ограниченных `<?php` и `?>`, выводится в браузер непосредственно, т. е. воспринимается, как вызов оператора `echo` (последняя аналогия очень точна, и мы остановимся на ней чуть позже).

Нетрудно догадаться, что часть строки после `//` является комментарием и на программу никак не влияет. Однострочные комментарии также можно предварять и символом `#` вместо `//`, как мы можем это увидеть в примере. Комментарии еще бывают и такие:

```

/*
это комментарий
...и еще одна строка
*/

```

То есть, комментарии могут быть однострочными и многострочными.

Однако в некоторых реализациях PHP многострочные комментарии почему-то вступают в конфликт с "русскими" буквами, которые могут находиться между ними. А именно, появляются бессмысленные сообщения о синтаксических ошибках, причем совершенно не в том месте. Почему так происходит, неясно: видимо, ошибка в PHP. Насчет комментариев и контроля ошибок мы еще поговорим, а пока вот вам совет: никогда не пользуйтесь многострочными комментариями в PHP.

А пока давайте лучше посмотрим, что происходит дальше. Вот строка:
`$dat=date("d.m y");`

Делает она следующее: переменной с именем `$dat` (заметьте, что абсолютно все переменные в PHP должны начинаться со знака `$`, потому что "так проще для интерпретации") присваивается значение, которое вернула функция `date()`. Итак, мы видим, что в PHP, во-первых, нет необходимости явно описывать переменные, а во-вторых, нигде не указывается их тип (про типы мы еще поговорим чуть позже). Интерпретатор сам решает, что, где и какого типа. А насчет функции `date()`... Можно заметить, что у нее задается один параметр, который определяет формат результата. Например, в нашем случае это будет строка вида "01.01 09".

В конце каждого оператора должна стоять точка с запятой. Иными словами, вы обязаны ставить точку с запятой перед `else` в конструкции `if-else`, но не должны после заголовка функции.

На следующей строке мы опять видим комментарии, а дальше — еще один оператор, похожий на ранее описанный. Он присваивает переменной `$tm` текущее время в формате "часы:минуты:секунды", опять же при помощи вызова `date()`. Все возможности этой полезной функции будут подробно описаны далее.

Далее следуют операторы `echo`, выводящие текстовые строки и нашу дату и время.

Рассмотрим один из них:

```
echo "Текущая дата: $dat года \n";
```

Заметьте: то, что любая переменная должна начинаться с символа `$`, позволяет интерпретатору вставить ее прямо в строку символов на место `$dat` (конечно, в любую строку, а не только в параметры `echo`). Разумеется, можно было бы написать и так (поскольку конструкция `echo` не ограничена по числу параметров):

```
echo "Текущая дата: ", $dat, " года \n";
```

или даже так:

```
echo "Текущая дата: ".$dat." года \n";
```

так как для слияния строк используется операция `.` (к этому придется пока привыкнуть).

Кстати говоря, на вопрос, почему для конкатенации строк применяется точка а не, скажем, плюс `+`, довольно легко ответить примером:

```
<?php
$a="100";
$b="200";
echo "$a+$b <br>"; // выведет "100+200"
echo "$a.$b"; // выведет "100.200"
?>
```

В окне веб-браузера это будет выглядеть [ТАК](#).

Итак, мы видим, что плюс используется именно как числовой оператор, а точка — как строковой. Все нюансы применения операторов мы рассмотрим далее.

Еще один пример "внедрения" переменных непосредственно в строку:

```
$path="c:/windows"; $name="win"; $ext="com";
FullPath="$path\".$name.$ext";
```

Последнее выглядит явно изящнее, чем:

```
$path="c:/windows"; $name="win"; $ext="com";
$FullPath=$path."\".$name.".".$ext;.
```

Ну вот, мы почти подобрались к сердцу нашего сценария — "уникальному" алгоритму поиска квадратов и кубов первых 5 натуральных чисел. Выглядит он так:

```
for($i=1; $i<=5; $i++)
{ echo "<li>$i в квадрате = ".$i*$i);
echo ", $i в кубе = ".$i*$i*$i)."\n";
}
```

В первой строке находится определение цикла for (счетчик \$i, которому присваивается начальное значение 1, инкрементируется на единицу на каждом шаге, пока не достигнет пяти). Затем следует блок, выполняющий вывод одной пары "квадрат-куб". Вывод сделан в две строки.

Несколько операторов можно сделать одним сложным оператором, заключив их в фигурные скобки, как это сделано выше.

Наконец, после всего этого расположен закрывающий тэг PHP ?>, а дальше — опять обычные HTML-тэги, завершающие нашу страничку.

Результат работы сценария.

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Изучаем PHP</title>
</head>

<body>
<h1>Здравствуйтесь!</h1>
Текущая дата: 01.01. 09 года<br>
Текущее время: 04:34:16<br>
А вот квадраты и кубы первых 5 натуральных чисел:<br>
<li>1 в квадрате = 1, 1 в кубе = 1
<li>2 в квадрате = 4, 2 в кубе = 8
<li>3 в квадрате = 9, 3 в кубе = 27
<li>4 в квадрате = 16, 4 в кубе = 64
<li>5 в квадрате = 25, 5 в кубе = 125
</body>
</html>
```

Как видно, выходные данные сценария скомбинировались с текстом, расположенным вне скобок <?php и ?>. В этом-то и заключена основная сила PHP: в легком встраивании кода в тело документа.

ЗАДАНИЕ.

1. Создайте файл 1-1.php, содержащий 5 разных переменных, присвойте переменным значения разного типа. Используя `gettype()` выведите тип каждой переменной.

2. Создайте файл 1-2.php, содержащий 2 переменные числового типа. Произведите над переменными произвольное арифметическое действие и выведите его результат.

3. Создайте файл 1-3.php, содержащий 2 переменные строкового типа. Инициализируйте переменные произвольным текстом. С помощью конкатенации объедините содержимое переменных и выведите результат.

4. Создайте файл 1-4.php, содержащий 2 переменные с одинаковым типом значений. Используя тернарный оператор сравнения проведите исследование на возвращаемые результаты.

Использование основных операторов PHP

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Приведение типов

1. Создать переменную `$var` и последовательно присвоить ей значения имеющие следующие типы:

- boolean

```
$var = false;
echo gettype($var);
```

- integer

```
$var = 11;
echo gettype($var);
```

- float

```
$var = 11.2;
echo gettype($var);
```

- string

```
$var = "String value";
echo gettype($var);
```

2. Использование функций приведения типов. Запустить и прокомментировать результат выполнения следующих функций:

- `intval()`

```
○ echo intval(42) . " <hr>";
○ echo intval(4.2) . " <hr>";
○ echo intval('42') . " <hr>";
○ echo intval('+42') . " <hr>";
○ echo intval('-42') . " <hr>";
○ echo intval(042) . " <hr>";
○ echo intval('042') . " <hr>";
```

- floatval()

```

○ echo floatval('122') ." <hr>";
○ echo floatval('The') ." <hr>";
○ echo floatval('122.34343') ." <hr>";
○ echo floatval('122.34343The') ." <hr>";
○ echo floatval('The122.34343') ." <hr>";
○ echo floatval('122.The34343') ." <hr>";

```

- floatval()

```

○ echo floatval('122') ." <hr>";
○ echo floatval('The') ." <hr>";
○ echo floatval('122.34343') ." <hr>";
○ echo floatval('122.34343The') ." <hr>";
○ echo floatval('The122.34343') ." <hr>";
○ echo floatval('122.The34343') ." <hr>";

```

- strval()

```

○ echo strval(122) ." <hr>";
○ echo strval(122.01) ." <hr>";
○ echo strval(0x122) ." <hr>";
○ echo strval(0122) ." <hr>";

```

- settype()

```

$foo = "5bar"; // string
$bar = true; // boolean
settype($foo, "integer"); // $foo is now 5 (integer)
settype($bar, "string"); // $bar is now "1" (string)

```

Операции с целыми числами

1. Выполните следующие операции с целыми числами

```

$a = 3;
$b = 2;
echo '<br>$a=' . $a;
echo '<br>$b=' . $b;

$c = $a+$b;
echo '<br>$a+$b=' . $c;

$c = $a-$b;
echo '<br>$a-$b=' . $c;

$c = $a*$b;
echo '<br>$a*$b=' . $c;

$c = $a/$b;
echo '<br>$a/$b=' . $c;

```

Операции со строками

1. Запустить и прокомментировать результат выполнения

```

$expand      =      "EXPAND";
$either      =      "EITHER";

echo 'это простая строка';
echo 'Также вы можете вставлять в строки
символ новой строки таким образом,
поскольку это нормально';
echo 'Однажды Арнольд сказал: "I\'ll be back"';
echo 'Вы удалили C:\\*.??';
echo 'Вы удалили C:/*.??';
echo 'Это не вставит: \n новую строку';
echo 'Переменные $expand также $either не подставляются';
echo "Переменные $expand также $either подставляются. Почему?";
echo $expand+ $either;

```

Приоритет операторов и управление им

1. Вывести результат выполнения следующих выражений:

```

1 + 5 * 3
(1 + 5) * 3
(12 + 13) - (4 + 6)
12 + 13 - 4 + 6

```

Пример желаемого вывода:

```
1 + 5 * 3 = 16
```

Операции с логическими переменными

- Создайте переменную \$age
- Присвойте переменной \$age произвольное числовое значение
- Напишите конструкцию if, которая выводит фразу: "Вам ещё работать и работать" при условии, что значение переменной \$age попадает в диапазон чисел от 18 до 59 (включительно)
- Расширьте конструкцию if, выводя фразу: "Вам пора на пенсию" при условии, что значение переменной \$age больше 59
- Расширьте конструкцию if-else, выводя фразу: "Вам ещё рано работать" при условии, что значение переменной \$age попадает в диапазон чисел от 1 до 17 (включительно)
- Дополните конструкцию if-elseif, выводя фразу: "Неизвестный возраст" при условии, что значение переменной \$age не попадает в вышеописанные диапазоны чисел

Задания для самостоятельного выполнения (все)

1. Используя условный переход, выведите сообщение «Счастливчик!» если \$age попадает в диапазон между 18 и 35. Если значение иное, выведите «Не повезло». Расширьте предыдущую конструкцию сообщением «Слишком молод», если \$age в диапазоне между 1 и 17.

2. Используя циклы, сформируйте массив четных чисел из диапазона от 1 до 100. Выводя массив на экран, исключите из вывода все числа, которые не делятся на 5.

3. Создайте массив со следующими элементами: Name, Address, Phone, Mail и заполните его. С помощью цикла foreach осуществите форматированный вывод массива в виде: «элемент: значение».

4. while

Вывести последовательно числа от 1990 до 2007 используя цикл while.

5. do-while

Вывести последовательно числа от 1990 до 2007 используя цикл do while.

6. for

Вывести последовательно числа от 1990 до 2007 используя цикл for.

7. break

Вывести последовательно числа от 1990 до 2007 используя цикл while. Прервать вывод на 1995 году.

8. continue

Вывести последовательно числа от 1990 до 2007 используя цикл while. Не выводить года с 1994 по 1997.

9. switch

Задать значение переменной \$name. Произвести проверку переменной \$name на имена "John", "Bill", "Sam". Также, в случае отрицательного результата вывести "Приветствую Незнакомец".

Лабораторная работа №9 Обработка строковых данных на PHP

Цель работы: Изучить функции обработки строковых данных

Порядок выполнения работы.

Изучить теоретические сведения.

Выполнить задание к лабораторной работе в соответствии с вариантом.

Оформить отчет.

Требования к отчету.

Цель работы.

Постановка задачи.

Текст программы.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Строка в PHP - это набор символов любой длины. В отличие от Си, строки могут содержать в себе также и нулевые символы, что никак не повлияет на программу. Иными словами, строки можно использовать для хранения бинарных данных. Длина строки ограничена только размером свободой оперативной памяти.

В PHP символ это то же самое, что и байт, это значит, что возможно ровно 256 различных символов. Это также означает, что PHP не имеет встроенной поддержки Unicode. Некоторую поддержку Unicode обеспечивают функции [utf8_encode\(\)](#) и [utf8_decode\(\)](#).

Строка легко может быть обработана при помощи стандартных функций, можно также непосредственно обратиться к любому ее символу.

Простой пример строковой переменной:

```
<?
$a = "Это просто текст, записанный в строковую переменную";
echo $a; //Выводит 'Это просто текст, записанный в строковую переменную'
?>
```

А теперь подробно разберем синтаксис типа данных **string**.

Синтаксис типа string (строка)

Строка может быть определена тремя различными способами.

- [одинарными кавычками](#)
- [двойными кавычками](#)
- [heredoc-синтаксисом](#)

Определение строк одинарными кавычками:

Простейший способ определить строку - это заключить ее в одинарные кавычки (символ `'`).

Чтобы использовать одинарную кавычку внутри строки, как и во многих других языках, ее необходимо предварить символом обратной косой черты (`\`), т. е. экранировать ее. Если обратная косая черта должна идти перед одинарной кавычкой либо быть в конце строки, вам необходимо продублировать ее. Обратите внимание, что если вы попытаетесь экранировать любой другой символ, обратная косая черта также будет

напечатана! Так что, как правило, нет необходимости экранировать саму обратную косую черту.

В отличие от двух других синтаксисов, переменные и экранирующие последовательности для специальных символов, встречающиеся в строках, заключенных в одинарные кавычки, **не** обрабатываются.

Приведем пример использования одинарных кавычек:

```
<?php
echo 'это простая строка';

echo 'Также вы можете вставлять в строки
символ новой строки таким образом,
поскольку это нормально';

// Выведет: Однажды Арнольд сказал: "I'll be back"
echo 'Однажды Арнольд сказал: "I\'ll be back"';

// Выведет: Вы удалили C:\*.*?
echo 'Вы удалили C:\\*.*?';

// Выведет: Вы удалили C:\*.*?
echo 'Вы удалили C:\*.*?';

// Выведет: Это не вставит: \n новую строку
echo 'Это не вставит: \n новую строку';

// Выведет: Переменные $exprand также $either не подставляются
echo 'Переменные $exprand также $either не подставляются';
?>
```

Определение строк двойными кавычками:

Если строка заключена в двойные кавычки ("), PHP распознает большее количество управляющих последовательностей для специальных символов:

Таблица управляющих последовательностей:

Последовательность	Значение
<code>\n</code>	новая строка (LF или 0x0A (10) в ASCII)
<code>\r</code>	возврат каретки (CR или 0x0D (13) в ASCII)
<code>\t</code>	горизонтальная табуляция (HT или 0x09 (9) в ASCII)
<code>\\</code>	обратная косая черта
<code>\\$</code>	знак доллара
<code>\"</code>	двойная кавычка
<code>[0-7]{1,3}</code>	последовательность символов, соответствующая регулярному выражению, символ в восьмеричной системе счисления

Последовательность	Значение
<code>\x{0-9A-Fa-f}{1,2}</code>	последовательность символов, соответствующая регулярному выражению, символ в шестнадцатеричной системе счисления

Еще раз напомним, что если вы захотите мнемонизировать любой другой символ, обратная косая черта также будет напечатана!

Самым важным свойством строк в двойных кавычках является обработка переменных. Смотрите более подробно: [обработка строк](#).

Определение строк heredoc-синтаксисом:

Другой способ определения строк - это использование **heredoc-синтаксиса** ("<<<"). После <<< необходимо указать идентификатор, затем идет строка, а потом этот же идентификатор, закрывающий вставку.

Закрывающий идентификатор должен начинаться в первом столбце строки. Кроме того, идентификатор должен соответствовать тем же правилам именования, что и все остальные метки в РНР: содержать только буквенно-цифровые символы и знак подчеркивания, и должен начинаться с нецифры или знака подчеркивания.

Внимание! Очень важно отметить, что строка с закрывающим идентификатором не содержит других символов, за исключением, возможно, точки с запятой (;). Это означает, что идентификатор не должен вводиться с отступом и что не может быть никаких пробелов или знаков табуляции до или после точки с запятой. Важно также понимать, что первым символом перед закрывающим идентификатором должен быть символ новой строки, определенный в вашей операционной системе. Например, на Windows® это `\r`.

Если это правило нарушено и закрывающий идентификатор не является "чистым", считается, что закрывающий идентификатор отсутствует и РНР продолжит его поиск дальше. Если в этом случае верный закрывающий идентификатор так и не будет найден, то это вызовет ошибку в обработке с номером строки в конце скрипта.

Heredoc-текст ведет себя так же, как и строка в двойных кавычках, при этом их не имея. Это означает, что вам нет необходимости экранировать кавычки в heredoc, но вы по-прежнему можете использовать вышеперечисленные управляющие последовательности. Переменные обрабатываются, но с применением сложных переменных внутри heredoc нужно быть также внимательным, как и при работе со строками.

Пример определения heredoc-строки:

```
<?php
$str = <<<EOD
Пример строки,
охватывающей несколько строчек,
с использованием heredoc-синтаксиса.
EOD;
```

```
/* Более сложный пример с переменными. */
```

```

class foo
{
    var $foo;
    var $bar;

    function foo()
    {
        $this->foo = 'Foo';
        $this->bar = array('Bar1', 'Bar2', 'Bar3');
    }
}

$foo = new foo();
$name = 'МоеИмя';

echo <<<ЕОТ
Меня зовут "$name". Я печатаю $foo->foo.
Теперь я вывожу {$foo->bar[1]}.
Это должно вывести заглавную букву 'A': \x41
ЕОТ;
?>

```

Если строка определяется в двойных кавычках, либо при помощи heredoc, переменные внутри нее обрабатываются.

Существует два типа синтаксиса: [простой](#) и [сложный](#). Простой синтаксис более легок и удобен. Он дает возможность обработки переменной, значения массива ([array](#)) или свойства объекта ([object](#)).

Сложный синтаксис был введен в РНР 4 и может быть распознан по фигурным скобкам, окружающим выражение.

Простой синтаксис

Если интерпретатор встречает знак доллара (\$), он захватывает так много символов, сколько возможно, чтобы сформировать правильное имя переменной. Если вы хотите точно определить конец имени, заключайте имя переменной в фигурные скобки.

```

<?php
$beer = 'Heineken';
echo "$beer's taste is great"; // работает, "'" это неверный символ для имени
    переменной
echo "He drank some $beers"; // не работает, 's' это верный символ для имен
    и переменной
echo "He drank some ${beer}s"; // работает
echo "He drank some {$beer}s"; // работает
?>

```

Точно также могут быть обработаны элемент массива ([array](#)) или свойство объекта ([object](#)). В индексах массива закрывающая квадратная скобка (]) обозначает конец определения индекса. Для свойств объекта применяются те же правила, что и для простых переменных, хотя с ними невозможен трюк, как с переменными.

```

<?php
// Эти примеры специфически об использовании массивов внутри
// строк. Вне строк всегда заключайте строковые ключи вашего
// массива в кавычки и не используйте вне строк {скобки}.

```

```
// Давайте покажем все ошибки
error_reporting(E_ALL);

$fruits = array('strawberry' => 'red', 'banana' => 'yellow');

// Работает, но заметьте, что вне кавычек строки это работает по-другому
echo "A banana is $fruits[banana].";

//Работает
echo "A banana is {$fruits['banana']}.";

// Работает, но PHP, как описано ниже, сначала ищет
// константу banana.
echo "A banana is {$fruits[banana]}.";

// Не работает, используйте фигурные скобки. Это вызовет ошибку обработки.
echo "A banana is $fruits['banana'].";

// Работает
echo "A banana is " . $fruits['banana'] . ".";

// Работает
echo "This square is $square->width meters broad.";

// Не работает. Для решения см. сложный синтаксис.
echo "This square is $square->width00 centimeters broad.";
?>
```

Для более сложных задач вы можете использовать сложный синтаксис.

Сложный (фигурный) синтаксис

Данный синтаксис называется сложным не потому, что труден в понимании, а потому что позволяет использовать сложные выражения.

Фактически, вы можете включить любое значение, находящееся в пространстве имени в строке с этим синтаксисом. Вы просто записываете выражение таким же образом, как и вне строки, а затем заключаете его в { и }. Поскольку вы не можете экранировать '{', этот синтаксис будет распознаваться только когда \$ следует непосредственно за {. (Используйте "{\$}" или "\{\$" чтобы отобразить "{\$}"). Несколько поясняющих примеров:

```
<?php
// Давайте покажем все ошибки
error_reporting(E_ALL);

$great = 'fantastic';

// Не работает, выведет: This is { fantastic}
echo "This is { $great}";

// Работает, выведет: This is fantastic
echo "This is {$great}";
echo "This is ${great}";
```

```

// Работает
echo "Этот квадрат шириной {$square->width}00 сантиметров.";

// Работает
echo "Это работает: {$arr[4][3]}";

// Это неверно по той же причине, что и $foo[bar] неверно вне
// строки. Другими словами, это по-прежнему будет работать,
// но поскольку PHP сначала ищет константу foo, это вызовет
// ошибку уровня E_NOTICE (неопределенная константа).
echo "Это неправильно: {$arr[foo][3]}";

// Работает. При использовании многомерных массивов, внутри
// строк всегда используйте фигурные скобки
echo "Это работает: {$arr['foo'][3]}";

// Работает.
echo "Это работает: " . $arr['foo'][3];

echo "Вы даже можете записать {$obj->values[3]->name}";

echo "Это значение переменной по имени $name: {${$name}}";
?>

```

Доступ к символу в строке и его изменение:

Символы в строках можно использовать и модифицировать, определив их смещение относительно начала строки, начиная с нуля, в фигурных скобках после строки. Приведем примеры:

```

<?php
// Получение первого символа строки
$str = 'Это тест.';
$first = $str{0};

// Получение третьего символа строки
$third = $str{2};

// Получение последнего символа строки
$str = 'Это все еще тест.';
$last = $str{strlen($str)-1};

// Изменение последнего символа строки
$str = 'Посмотри на море';
$str{strlen($str)-1} = 'я';

?>

```

Строковые функции и операторы

Строковые операторы

Конкатенация строк:

В различных языках программирования используются различные операторы конкатенации (объединения) строк. Например, в Pascal используется оператор "+". Использование в PHP оператора "+" для

конкатенации строк некорректно: если строки содержат числа, то вместо объединения строк будет выполнена операция сложения двух чисел.

В PHP есть два оператора, выполняющие конкатенацию.

Первый - оператор конкатенации ('.'), который возвращает объединение левого и правого аргумента.

Второй - оператор присвоения с конкатенацией, который присоединяет правый аргумент к левому.

Приведем конкретный пример:

```
<?php
$a = "Hello ";
$b = $a . "World!"; // $b содержит строку "Hello World!" - Это конкатенация

$a = "Hello ";
$a .= "World!"; // $a содержит строку "Hello World!" - Это присвоение с
конкатенацией
?>
```

Операторы сравнения строк

Для сравнения строк не рекомендуется использовать операторы сравнения == и !=, поскольку они требуют преобразования типов. Пример:

```
<?php
$x=0;
$y=1;
if ($x == "") echo "<p>x - пустая строка</p>";
if ($y == "") echo "<p>y - пустая строка</p>";
// Выводит:
// x - пустая строка
?>
```

Данный скрипт сообщает нам, что \$x - пустая строка. Это связано с тем, что пустая строка ("") трактуется прежде всего как 0, а только затем - как "пусто". В PHP операнды сравниваются, как строки, только в том случае, если оба они - строки. В противном случае они сравниваются как числа. При этом любая строка, которую PHP не удастся перевести в число (в том числе и пустая строка), будет восприниматься как 0.

Примеры сравнения строк:

```
<?php
$x="Строка";
$y="Строка";
$z="Строчка";
if ($x == $z) echo "<p>Строка X равна строке Z</p>";
if ($x == $y) echo "<p>Строка X равна строке Y</p>";
if ($x != $z) echo "<p>Строка X НЕ равна строке Z</p>";
// Выводит:
// Строка X равна строке Y
// Строка X НЕ равна строке Z
?>
```

Чтобы избежать путаницы и преобразования типов, рекомендуется пользоваться оператором эквивалентности при сравнении строк. Оператор эквивалентности позволяет всегда корректно сравнивать строки, поскольку сравнивает величины и по значению, и по типу:

```

<?php
$x="Строка";
$y="Строка";
$z="Строчка";
if ($x === $z) echo "<p>Строка X равна строке Z</p>";
if ($x === $y) echo "<p>Строка X равна строке Y</p>";
if ($x !== $z) echo "<p>Строка X НЕ равна строке Z</p>";
// Выводит:
// Строка X равна строке Y
// Строка X НЕ равна строке Z
?>

```

Базовые строковые функции

strlen(string \$st)

Одна из наиболее полезных функций. Возвращает просто длину строки, т. е., сколько символов содержится в `$st`. Строка может содержать любые символы, в том числе и с нулевым кодом (что запрещено в Си). Пример:

```

$x = "Hello!";
echo strlen($x); // Выводит 6

```

strpos(string \$where, string \$what, int \$fromwhere=0)

Пытается найти в строке `$where` подстроку (то есть последовательность символов) `$what` и в случае успеха возвращает позицию (индекс) этой подстроки в строке. Необязательный параметр `$fromwhere` можно задавать, если поиск нужно вести не с начала строки `$from`, а с какой-то другой позиции. В этом случае следует эту позицию передать в `$fromwhere`. Если подстроку найти не удалось, функция возвращает `false`. Однако будьте внимательны, проверяя результат вызова `strpos()` на `false` — используйте для этого только оператор `===`. Пример:

```

echo strpos("Hello", "el"); // Выводит 1

```

И еще пример:

```

if (strpos("Norway", "rwa") !== false) echo "Строка rwa есть в Norway";
// При сравнении используйте операторы тождественных сравнений (===) (!==) чтобы избежать проблем с определением типов

```

substr(string \$str, int \$start [,int \$length])

Данная функция тоже востребуется очень часто. Ее назначение — возвращать участок строки `$str`, начиная с позиции `$start` и длиной `$length`. Если `$length` не задана, то подразумевается подстрока от `$start` до конца строки `$str`. Если `$start` больше, чем длина строки, или же значение `$length` равно нулю, то возвращается пустая подстрока. Однако эта функция может делать и еще довольно полезные вещи. К примеру, если мы передадим в `$start` отрицательное число, то будет считаться, что это число является индексом подстроки, но только отсчитываемым от конца `$str` (например, `-1` означает "начиная с последнего символа строки"). Параметр `$length`, если он задан, тоже может быть отрицательным. В этом случае последним символом возвращенной подстроки будет символ из `$str` с индексом `$length`, определяемым от конца строки. Примеры:


```

$str = "Programmer";
echo substr($str,0,2); // Выводит Pr
echo substr($str,-3,3); // Выводит mer

```

strcmp(string \$str1, string \$str2)

Сравнивает две строки посимвольно (точнее, побайтово) и возвращает: 0, если строки полностью совпадают; -1, если строка `$str1` лексикографически меньше `$str2`; и 1, если, наоборот, `$str1` "больше" `$str2`. Так как сравнение идет побайтово, то регистр символов влияет на результаты сравнений.

strcasecmp(string \$str1, string \$str2)

То же самое, что и `strcmp()`, только при работе не учитывается регистр букв. Например, с точки зрения этой функции "ab" и "AB" равны.

Функции для работы с блоками текста

Перечисленные ниже функции чаще всего оказываются полезны, если нужно проводить однотипные операции с многострочными блоками текста, заданными в строковых переменных.

str_replace(string \$from, string \$to, string \$str)

Заменяет в строке `$str` все вхождения подстроки `$from` (с учетом регистра) на `$to` и возвращает результат. Исходная строка, переданная третьим параметром, при этом не меняется. Эта функция работает значительно быстрее, чем [ereg_replace\(\)](#), которая используется при работе с регулярными выражениями PHP, и ее часто используют, если нет необходимости в каких-то экзотических правилах поиска подстроки. Например, вот так мы можем заместить все символы перевода строки на их HTML эквивалент — тэг `
`:

```
$st=str_replace("\n","<br>\n",$str)
```

Как видим, то, что в строке `
\n` тоже есть символ перевода строки, никак не влияет на работу функции, т. е. функция производит лишь однократный проход по строке. Для решения описанной задачи также применима функция `nl2br()`, которая работает чуть быстрее.

string nl2br(string \$string)

Заменяет в строке все символы новой строки `\n` на `
\n` и возвращает результат. Исходная строка не изменяется. Обратите внимание на то, что символы `\r`, которые присутствуют в конце строки текстовых файлов Windows, этой функцией никак не учитываются, а потому остаются на старом месте.

WordWrap(string \$str, int \$width=75, string \$break="\n")

Эта функция, появившаяся в PHP4, оказывается невероятно полезной, например, при форматировании текста письма перед автоматической отправкой его адресату при помощи `mail()`. Она разбивает блок текста `$str` на несколько строк, завершаемых символами `$break`, так, чтобы на одной строке было не более `$width` букв. Разбиение происходит по границе слова, так что текст остается читаемым. Возвращается получившаяся строка с символами перевода строки, заданными в `$break`. Пример использования:

```

<?php
$str = "Это текст электронного письма, которое нужно будет отправить адресату
...";
// Разбиваем текст по 20 символов
$str = WordWrap ($str, 20, "<br>");
echo $str;
// Выводит:
/* Это текст
электронного письма,
которое нужно будет
отправить
адресату... */
?>

```

strip_tags (string \$str [, string \$allowable_tags])

Еще одна полезная функция для работы со строками. Эта функция удаляет из строки все тэги и возвращает результат. В параметре `$allowable_tags` можно передать тэги, которые не следует удалять из строки. Они должны перечисляться вплотную друг к другу. Примеры:

```

$stripped = strip_tags ($str); // Удаляет все html - теги из строки
(текста)
$stripped = strip_tags($str, "<head><title>"); // Удалит все html - теги,
кроме html - тегов <head> и <title>

```

Функции для работы с отдельными символами

Как и в других языках программирования, в PHP можно работать с символами строк отдельно.

Обратиться к любому символу строки можно по его индексу:

```

$str = "PHP";
echo $str[0]; // Выводит 'P'

```

chr(int \$code)

Данная функция возвращает строку, состоящую из символа с кодом `$code`. Пример:

```

echo chr(75); //Выводит K

```

ord(\$char)

Данная функция возвращает код символа `$char`. Вот пример:

```

echo ord('A'); // Выводит 65 - код буквы 'A'

```

Функции удаления пробелов

Иногда трудно даже представить, какими могут быть странными пользователи, если дать им в руки клавиатуру и попросить напечатать на ней какое-нибудь слово. Так как клавиша пробела — самая большая, то пользователи имеют обыкновение нажимать ее в самые невероятные моменты. Этому способствует также и тот факт, что символ с кодом 32, обозначающий пробел, как вы знаете, на экране не виден. Если программа не способна обработать описанную ситуацию, то она, в лучшем случае после тягостного молчания отобразит в браузере что-нибудь типа "неверные входные данные", а в худшем — сделает при этом что-нибудь необратимое.

Между тем, обезопасить себя от паразитных пробелов чрезвычайно просто, и разработчики PHP предоставляют нам для этого ряд специализированных функций. Не волнуйтесь о том, что их применение

замедляет программу. Эти функции работают с молниеносной скоростью, а главное, одинаково быстро, независимо от объема переданных им строк.

trim(string \$str)

Возвращает копию \$str, только с удаленными ведущими и концевыми пробельными символами. Под пробельными символами я здесь и далее подразумеваю: пробел " ", символ перевода строки \n, символ возврата каретки \r и символ табуляции \t. Например, вызов trim(" test\n ") вернет строку "test". Эта функция используется очень широко. Старайтесь применять ее везде, где есть хоть малейшее подозрение на наличие ошибочных пробелов. Поскольку работает она очень быстро.

ltrim(string \$st)

То же, что и trim(), только удаляет исключительно ведущие пробелы, а концевые не трогает. Используется гораздо реже.

chop(string \$st)

Удаляет только концевые пробелы, ведущие не трогает.

Функции преобразования символов

Web-программирование — одна из тех областей, в которых постоянно приходится манипулировать строками: разрывать их, добавлять и удалять пробелы, перекодировать в разные кодировки, наконец, URL-кодировать и декодировать. В PHP реализовать все эти действия вручную, используя только уже описанные примитивы, просто невозможно из соображений быстродействия. Поэтому-то и существуют подобные встроенные функции.

strtr(string \$str, string \$from, string \$to)

Эта функция применяется не столь широко, но все-таки иногда она бывает довольно полезной. Она заменяет в строке \$str все символы, встречающиеся в \$from, на их "парные" (то есть расположенные в тех же позициях, что и во \$from) из \$to.

Следующие несколько функций предназначены для быстрого URL-кодирования и декодирования.

URL-кодирование необходимо для передачи данных через интернет. Например, такое кодирование целесообразно, если вы передаете русскоязычную информацию в качестве параметра скрипта. Также подобное кодирование можно выполнить и для файла, чтобы не возникало коллизий из-за отсутствия поддержки 8-битных кодировок некоторыми серверами. Вот эти функции:

urlencode(string \$str)

Функция URL-кодирует строку \$str и возвращает результат. Эту функцию удобно применять, если вы, например, хотите динамически сформировать ссылку на какой-то сценарий, но не уверены, что его параметры содержат только алфавитно-цифровые символы. В этом случае воспользуйтесь функцией так:

```
echo "<a href=/script.php?param=".urlencode($UserData);
```

Теперь, даже если переменная `$UserData` включает символы `=`, `&` или даже пробелы, все равно сценарию будут переданы корректные данные.

UrlDecode(string \$st)

Производит URL-декодирование строки. В принципе, используется значительно реже, чем `urlencode()`, потому что PHP и так умеет перекодировать входные данные автоматически.

Rawurlencode(string \$st)

Почти полностью аналогична `urlencode()`, но только пробелы не преобразуются в `+`, как это делается при передаче данных из формы, а воспринимаются как обычные неалфавитно-цифровые символы. Впрочем, этот метод не порождает никаких дополнительных несовместимостей в коде.

Rawurldecode(string \$st)

Аналогична `urldecode()`, но не воспринимает `+` как пробел.

htmlspecialchars(string \$str)

Это функция, которая обычно используется в комбинации с `echo`. Основное ее назначение - гарантировать, что в выводимой строке ни один участок не будет воспринят как тэг.

Заменяет в строке некоторые символы (такие как амперсant, кавычки и знаки "больше" и "меньше") на их HTML-эквиваленты, так, чтобы они выглядели на странице "самими собой". Самое типичное применение этой функции — формирование параметра `value` в различных элементах формы, чтобы не было никаких проблем с кавычками, или же вывод сообщения в гостевой книге, если вставлять тэги пользователю запрещено.

htmlspecialchars_decode(string \$str)

Заменяет в строке `$str` некоторые предваренные слэшем символы на их однокодовые эквиваленты. Это относится к следующим символам: `"`, `'`, `\` и никаким другим.

htmlspecialchars_decode(string \$str)

Вставляет слэши только перед следующими символами: `'`, `"` и `\`. Функцию очень удобно использовать при вызове [eval\(\)](#) (эта функция выполняет строку, переданную ей в параметрах, так, как будто имеет дело с небольшой PHP-программой).

Функции изменения регистра

Довольно часто нам приходится переводить какие-то строки, скажем, в верхний регистр, т. е. делать все прописные буквы в строке заглавными. В принципе, для этой цели можно было бы воспользоваться функцией `strtoupper()`, рассмотренной выше, но она все же будет работать не так быстро, как нам иногда хотелось бы. В PHP есть функции, которые предназначены специально для таких нужд. Вот они:

strtolower(string \$str)

Преобразует строку в нижний регистр. Возвращает результат перевода.

Надо заметить, что при неправильной настройке локали (это набор правил по переводу символов из одного регистра в другой, переводу даты и времени, денежных единиц и т. д.) функция будет выдавать неправильные результаты при работе с буквами кириллицы.

Возможно, в несложных программах, а также если нет уверенности в поддержке соответствующей локали операционной системой, проще будет воспользоваться "ручным" преобразованием символов, задействуя функцию `strtr()`:

```
$st=strtr($st, "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ",
"абвгдеёжзийклмнопрстуфхцчшщъыьэюя");
```

Главное достоинство данного способа — то, что в случае проблем с кодировкой для восстановления работоспособности сценария вам придется всего лишь преобразовать его в ту же кодировку, в которой у вас хранятся документы на сервере.

strtoupper(string \$str)

Переводит строку в верхний регистр. Возвращает результат преобразования. Эта функции также прекрасно работает со строками, составленными из латиницы, но с кириллицей может возникнуть все та же проблема.

Установка локали (локальных настроек)

Локалью будем называть совокупность локальных настроек системы, таких как формат даты и времени, язык, кодировка.

Настройки локали сильно зависят от операционной системы.

Для установки локали используется функция `SetLocale()`:

SetLocale(string \$category, string \$locale)

Функция устанавливает текущую локаль, с которой будут работать функции преобразования регистра символов, вывода даты-времени и т.д. Вообще говоря, для каждой категории функций локаль определяется отдельно и выглядит по-разному. То, какую именно категорию функций затронет вызов `SetLocale()`, задается в параметре `$category`. Он может принимать следующие строковые значения:

`LC_STYPE` — активизирует указанную локаль для функций перевода в верх-

ний/нижний регистры;

`LC_NUMERIC` — активизирует локаль для функций форматирования дробных чи-

сел — а именно, задает разделитель целой и дробной части в числах;

`LC_TIME` — задает формат вывода даты и времени по умолчанию;

`LC_ALL` — устанавливает все вышеперечисленные режимы.

Теперь о параметре `$locale`. Как известно, каждая локаль, установленная в системе, имеет свое уникальное имя, по которому к ней можно обратиться. Именно оно и фиксируется в этом параметре. Однако, есть

два важных исключения из этого правила. Во-первых, если величина `$locale` равна пустой строке "", то устанавливается та локаль, которая указана в глобальной переменной окружения с именем, совпадающим с именем категории `$category` (или `LANG` — она практически всегда присутствует в Unix). Во вторых, если в этом параметре передается 0, то новая локаль не устанавливается, а просто возвращается имя текущей локали для указанного режима.

К сожалению, имена локалей задаются при настройке операционной системы, и для них, по-видимому, не существует стандартов. Выясните у своего хостинг-провайдера, как называются локали для разных кодировок русских символов. Но, если следующий фрагмент работает у вашего хостинг-провайдера, это не означает, что он заработает, например, под Windows:

```
setlocale('LC_CTYPE','ru_SU.KOI8-R');
```

Здесь вызов устанавливает таблицу замены регистра букв в соответствии с кодировкой KOI8-R.

По правде говоря, локаль - вещь довольно непредсказуемая и довольно плохо переносимая между операционными системами. Так что, если ваш сценарий не очень велик, задумайтесь: возможно, лучше будет искать обходной путь, например, используя `strtr()`, а не рассчитывать на локаль.

Функции преобразования кодировок

Часто встречается ситуация, когда нам требуется преобразовать строку из одной кодировки кириллицы в другую. Например, мы в программе сменили локаль: была кодировка `windows`, а стала — `KOI8-R`. Но строки-то остались по-прежнему в кодировке `WIN-1251`, а значит, для правильной работы с ними нам нужно их перекодировать в `KOI8-R`. Для этого и служит функция преобразования кодировок.

```
convert_cyr_string(string $str, char $from, char $to);
```

Функция переводит строку `$str` из кодировки `$from` в кодировку `$to`. Конечно, это имеет смысл только для строк, содержащих "русские" буквы, т. к. латиница во всех кодировках выглядит одинаково. Разумеется, кодировка `$from` должна совпадать с истинной кодировкой строки, иначе результат получится неверным. Значения `$from` и `$to` — один символ, определяющий кодировку:

```
k — koi8-r
w — windows-1251
i — iso8859-5
a — x-cp866
d — x-cp866
m — x-mac-cyrillic
```

Функция работает достаточно быстро, так что ее вполне можно применять, скажем, для перекодировки писем в нужную форму перед их отправкой по электронной почте.

Функции форматных преобразований строк

Как мы знаем, переменные в строках PHP интерполируются, поэтому практически всегда задача "смешивания" текста со значениями переменных не является проблемой. Например, мы можем спокойно написать что-то вроде:

```
echo "Привет, $name! Вам $age лет.";
```

В Си для аналогичных целей используется следующий код:

```
printf("Привет, %s! Вам %s лет", name, age);
```

Язык PHP также поддерживает ряд функций, использующих такой же синтаксис, как и их Си -эквиваленты. Бывают случаи, когда их применение дает наиболее красивое и лаконичное решение, хотя это и случается довольно нечасто.

sprintf(string \$format [, mixed args, ...])

Эта функция — аналог функции `sprintf()` в Си. Она возвращает строку, составленную на основе строки форматирования, содержащей некоторые специальные символы, которые будут впоследствии заменены на значения соответствующих переменных из списка аргументов. Строка форматирования `$format` может включать в себя команды форматирования, предваренные символом `%`. Все остальные символы копируются в выходную строку как есть. Каждый спецификатор формата (то есть, символ `%` и следующие за ним команды) соответствует одному, и только одному параметру, указанному после параметра `$format`. Если же нужно поместить в текст `%` как обычный символ, необходимо его удвоить:

```
echo sprintf("The percentage was %d%%", $percentage);
```

Каждый спецификатор формата включает максимум пять элементов (в порядке их следования после символа `%`):

>>> Необязательный спецификатор размера поля, который указывает, сколько символов будет отведено под выводимую величину. В качестве символов-заполнителей (если значение имеет меньший размер, чем размер поля для его вывода) может использоваться пробел или `0`, по умолчанию подставляется пробел. Можно задать любой другой символ-наполнитель, если указать его в строке форматирования, предварив апострофом `'`.

>>> Опциональный спецификатор выравнивания, определяющий, будет результат выровнен по правому или по левому краю поля. По умолчанию производится выравнивание по правому краю, однако можно указать и левое выравнивание, задав символ `-` (минус).

>>> Необязательное число, определяющее размер поля для вывода величины. Если результат не будет в поле помещаться, то он "вылезет" за края этого поля, но не будет усечен.

>>> Необязательное число, предваренное точкой `.`, предписывающее, сколько знаков после запятой будет в результирующей строке. Этот спецификатор учитывается только в том случае, если происходит вывод числа с плавающей точкой, в противном случае он игнорируется.

>>> Наконец, обязательный (заметьте — единственный обязательный!) спецификатор типа величины, которая будет помещена в выходную строку:

b — очередной аргумент из списка выводится как двоичное целое число;

c — выводится символ с указанным в аргументе кодом;

d — целое число;

f — число с плавающей точкой;

o — восьмеричное целое число;

s — строка символов;

x — шестнадцатеричное целое число с маленькими буквами *a-z*;

X — шестнадцатеричное число с большими буквами *A-Z*.

Вот как можно указать точность представления чисел с плавающей точкой:

```
$money1 = 68.75;
$money2 = 54.35;
$money = $money1 + $money2;
// echo $money выведет "123.1"...
$formatted = sprintf ("%01.2f", $money);
// echo $formatted выведет "123.10"!
```

Вот пример вывода целого числа, предваренного нужным количеством нулей:

```
$isodate=sprintf ("%04d-%02d-%02d", $year, $month, $day);
```

printf(string \$format [, mixed args, ...])

Делает то же самое, что и `sprintf()`, только результирующая строка не возвращается, а направляется в браузер пользователя.

number_format(float \$number, int \$decimals, string \$dec_point=".", string \$thousands_sep="");

Эта функция форматирует число с плавающей точкой с разделением его на триады с указанной точностью. Она может быть вызвана с двумя или четырьмя аргументами, но не с тремя! Параметр `$decimals` задает, сколько цифр после запятой должно быть у числа в выходной строке. Параметр `$dec_point` представляет собой разделитель целой и дробной частей, а параметр `$thousands_sep` — разделитель триад в числе (если указать на его месте пустую строку, то триады не отделяются друг от друга).

В PHP существует еще несколько функций для выполнения форматных преобразований, среди них - [sscanf\(\)](#) и [fscanf\(\)](#), которые часто применяются в Си. Однако в PHP их использование весьма ограничено: чаще всего для разбора строк оказывается гораздо выгоднее привлечь регулярные выражения или функцию [explode\(\)](#).

Хэш-функции

md5(string \$str)

Возвращает хэш-код строки `$str`, основанный на алгоритме корпорации RSA Data Security под названием "MD5 Message-Digest Algorithm". Хэш-код — это просто строка, практически уникальная для

каждой из строк `$str`. То есть вероятность того, что две разные строки, переданные в `$str`, дадут нам одинаковый хэш-код, стремится к нулю.

Если длина строки `$str` может достигать нескольких тысяч символов, то ее MD5-код занимает максимум 32 символа.

Для чего нужен хэш-код и, в частности, алгоритм MD5? Например, для проверки паролей на истинность.

Пусть, к примеру, у нас есть система со многими пользователями, каждый из которых имеет свой пароль. Можно, конечно, хранить все эти пароли в обычном виде, или зашифровать их каким-нибудь способом, но тогда велика вероятность того, что в один прекрасный день этот файл с паролями у вас украдут.

Сделаем так: в файле паролей будем хранить не сами пароли, а их (MD5) хэш-коды. При попытке какого либо пользователя войти в систему мы вычислим хэш-код только что введенного им пароля и сравним его с тем, который записан у нас в базе данных. Если коды совпадут, значит, все в порядке, а если нет — что ж, извините... Конечно, при вычислении хэш-кода какая-то часть информации о строке `$str` безвозвратно теряется. И именно это позволяет нам не опасаться, что злоумышленник, получивший файл паролей, сможет его когда-нибудь расшифровать. Ведь в нем нет самих паролей, нет даже их каких-то связных частей!

Пример использования алгоритма хеширования MD5:

```
<?php
$pass_a = "MySecret";
$pass_b = "MySecret";
// Выводим хеш-код строки MySecret ($pass_a) - исходный пароль
echo "<b>Хеш-код исходного пароля '$pass_a':</b><b style=\"color:green\">".md5($pass_a)."</b><br>";
// Выводим хеш-код строки MySecret ($pass_b) - верифицируемый пароль
echo "<b>Хеш-код верифицируемого пароля '$pass_b':</b><b style=\"color:green\">".md5($pass_b)."</b><br>";
// Сравниваем хеш-коды MD5 исходного и верифицируемого пароля
echo "<h3>Проверяем истинность введенного пароля:</h3>";
if (md5($pass_a)===md5($pass_b)) echo "<h3 style=\"color:green\">Пароль верный! (Хеш-коды совпадают)</h3>";
else echo "<h3 style=\"color:red\">Пароль неверный! (Хеш-коды не совпадают)</h3>";
// В данной ситуации выводит: Пароль верный! (Хеш-коды совпадают)
// Попробуйте изменить значение строки $pass_b :)
?>
```

crc32(string \$str)

Функция `crc32()` вычисляет 32-битную контрольную сумму строки `$str`. То есть, результат ее работы — 32 битное (4-байтовое) целое число. Эта функция работает гораздо быстрее `md5()`, но в то же время выдает гораздо менее надежные "хэш-коды" для строки.

Функции сброса буфера вывода flush()

Эта функция имеет очень и очень отдаленное отношение к работе со строками, но она еще дальше отстоит от других функций.

Начнем издалека: обычно при использовании echo данные не прямо сразу отправляются клиенту, а накапливаются в специальном буфере, чтобы потом транспортироваться большой "пачкой". Так получается быстрее.

Однако, иногда бывает нужно досрочно отправить все данные из буфера пользователю, например, если вы что-то выводите в реальном времени (так зачастую работают чаты). Вот тут-то вам и поможет функция flush(), которая отправляет содержимое буфера echo в браузер пользователя.

Задания к лабораторной работе (по вариантам)

Вариант 1

1. Дана строка 'Привет, мир!'. Сделайте из нее строку 'ПРИВЕТ МИР!'

2 Нарисуйте пирамиду, как показано на рисунке, только у вашей пирамиды должно столько рядов, чтобы последний элемент пирамидки состоял из одного символа. **Первый ряд пирамиды** должен храниться в переменной \$str (может иметь различное количество символов). *Подсказка: воспользуйтесь функциями [strlen](#) и [substr](#).*

```
xxxxxxxxxx
xxxxxxxxxx
xxxxxxxxxx
xxxxxxxxxx
xxxxxxxxxx
```

3. Дана строка 'Я-учу-PHP!'. Замените все дефисы на тег '!'

4. Дана строка 'я учу PHP!'. С помощью [функции explode](#) запишите **каждое слово** этой строки в **отдельный элемент массива**.

5. Дана строка 'html, php, js'. Удалите теги из этой строки.

6. Дана строка 'Мама мыла раму'. Узнайте количество букв 'а' и 'м', входящих в эту строку.

7. Проверьте, является ли слово **палиндромом** (одинаково читается во всех направлениях, примеры таких слов: **madam, otto, kayak, nun, level**)

Вариант 2

1. Дана строка 'PHP'. Сделайте из нее строку 'php'.

2 Дана строка 'я учу PHP!'. Вырежьте из нее слово 'учу' и слово 'PHP'.

3. Дана строка '31.12.2013'. Замените все точки на дефисы.

4. В переменной `$date` лежит дата в формате '31.12.2013'. Преобразуйте эту дату в формат '2013-12-31'
5. Дана строка 'html, php, js'. Выведите ее на экран 'как есть': то есть браузер не должен преобразовать в жирный.
6. Запишите в переменную `$str` длинный текст. Подсчитайте количество символов и количество слов в этом тексте.
7. Определите является ли фраза палиндромом. **Примеры:** 'Never odd or even', 'A man, a plan, a canal. Panama'. **Обратите внимание** на то, что при обратном чтении игнорируются пробелы, запятые, дефисы, тире и большие буквы (подсказка: значит сначала нужно привести строку к стандартному виду - удалить лишние символы, привести все к нижнему регистру).

Вариант 3

1. Дана строка 'LONDON'. Сделайте из нее строку 'London'
2. Дана переменная `$str`, в которой хранится какой-либо текст. Реализуйте **обрезание длинного текста** по следующему принципу: если количество символов этого текста больше заданного в переменной `$n`, то в переменную `$result` запишем первые `$n` символов строки `$str` и добавим в конец троеточие '...'. В противном случае в переменную `$result` запишем содержимое переменной `$str`.
3. Дана строка `$str`. Замените смайлики ':)', ':(', '^_^', которые встречаются в этой строке на соответствующие картинки ().
4. В переменной `$date` лежит дата в формате '2013-12-31'. Преобразуйте эту дату в формат '31.12.2013'.
5. Дана строка ' php '. Сделайте из нее 3 разных строки с помощью функций класса trim: 'php', ' php', 'php '.
6. Создайте массив гласных букв. С помощью этого массива подсчитайте количество гласных в строке `$str`. Результат представьте в виде **ассоциативного массива**, где **ключами** будут буквы, а **элементами** - их количество.
7. Нарисуйте пирамиду, как показано на рисунке, только у вашей пирамиды должно быть 9 рядов, а не 5. Решите задачу с помощью **одного цикла** и [функции str_repeat](#).

```
x
xx
xxx
xxxx
xxxxx
```

Вариант 4

1. Дана строка 'london is the capital of great britain'. Сделайте из нее строку 'London Is The Capital Of Great Britain'

2. Дана переменная `$password`, в которой хранится пароль пользователя. Если **количество символов** пароля больше 5-ти и меньше 10-ти, то выведите пользователю сообщение о том, что пароль подходит, иначе сообщение о том, что нужно придумать другой пароль.

3. Дана переменная `$str`, в которой хранится строка русского текста. Напишите скрипт, который запишет **транслит** этого текста в переменную `$strslit`.

4. Дан массив с элементами `'html', 'css', 'php', 'js'`. С помощью [функции `implode`](#) создайте строку из этих элементов, разделенных запятыми.

5. Дана строка `'html, php, js'`. Выведите ее на экран **'как есть'**: то есть браузер не должен преобразовать `` в жирный.

6. Дана строка `'1234567890'`. Разбейте ее на массив с элементами `'12', '34', '56', '78', '90'`.

7. Нарисуйте пирамиду, как показано на рисунке, только у вашей пирамиды должно быть 9 рядов, а не 5. Решите задачу с помощью **одного цикла** и [функции `str_repeat`](#).

```
1
22
333
4444
55555
```

Лабораторная работа №10 Обработка массивов в PHP

Цель работы: Получение навыков в работе с массивами на языке PHP

Порядок выполнения работы.

Изучить теоретические сведения.

Выполнить задание к лабораторной работе в соответствии с вариантом.

Оформить отчет.

Требования к отчету.

Цель работы.

Постановка задачи.

Текст программы.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Массивы

Массивы являются самым распространенным и простейшим способом группировки данных. Изначально массив определяется как индексированная

совокупность переменных одного типа. Отдельные переменные в массиве принято называть *элементами массива*. Однако PHP расширяет это определение, в массивах PHP можно хранить переменные произвольного типа, вплоть до массивов и объектов. Для того чтобы обратиться к элементу массива, необходимо указать индекс или ключ. В листинге 2.1 создается массив \$arr из двух элементов, первый содержит строку "Hello", второй — "world".

Листинг 2.1. Обращение к элементам массива

```
<?php
// Создаем массив $arr из двух элементов
$arr[] = "Hello ";
$arr[] = "world";
// Выводим второй элемент массива echo $arr[1]; // world
?>
```

Для того чтобы обратиться к отдельному элементу массива, необходимо в квадратных скобках указать *индекс элемента массива*. Так обращение к \$arr[0] запрашивает первый элемент массива, \$arr [1] — второй и т. д.

Массивы, к элементам которых следует обращаться через числовой индекс, называют *индексными*. Помимо индексного массива PHP поддерживает работу с *ассоциативными массивами*. К элементам таких массивов можно обращаться через строковый индекс, который принято называть *ключом*. В листинге 2.2 создается ассоциативный массив \$arr из двух элементов, обратиться к первому из которых можно по ключу first, а ко второму — по ключу second.

Листинг 2.2. Обращение к элементам ассоциативного массива

```
<?php
// Создаем массив $arr из двух элементов
$arr['first'] = "Hello";
$arr['second'] = "world";
// Выводим второй элемент массива echo $arr[1]; // world
?>
```

В последующих разделах мы подробно ознакомимся с тем, как эффективно организовать работу с массивами.

Создание массива

Для создания массивов PHP предоставляет множество способов, начиная от специализированных конструкций, заканчивая стандартными функциями. Как будет показано в следующих лабораторных работах, многие функции расширений возвращают результаты своей работы в виде массивов. В последующих подразделах будут рассмотрены базовые способы создания массивов, поддерживаемые ядром PHP.

Конструкция array()

Существует несколько способов создания массивов. Первый из них заключается в использовании конструкции `array()`, в круглых скобках которой через запятую перечисляются элементы массива. В листинге 2.3 создается массив `$arr` состоящий из трех элементов, пронумерованных от 0 до 2.

Листинг 2.3. Создание массива при помощи конструкции `array()`

```
<?php
    $arr = array("Hello ", "world", "!");
    echo $arr[0]; // Hello
    echo $arr[1]; // world echo $arr[2]; // !
?>
```

В листинге 2.3 каждый элемент массива выводится при помощи отдельной конструкции `echo`. При попытке вывода всего массива `$arr` в окно браузера вместо его содержимого будет выведена строка "Array". Для просмотра структуры и содержимого массива предусмотрена специальная функция `print_r()`. В листинге 2.4 с ее помощью выводится структура массива `$arr`. Для удобства восприятия вызов функции `print_r()` обрамляется HTML-тегами `<pre>` и `</pre>`, которые сохраняют структуру переносов и отступов при отображении результата в браузере.

Листинг 2.4. Вывод структуры массива при помощи функции `print_r()`

```
<?php
    $arr = array("Hello ", "world", "!");
    echo "<pre>";
    print_r($arr);
    echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 2.4 будет следующий дамп массива `$arr`:

```
Array
(
    [0] => Hello
    [1] => world
    [2] => !
)
```

Как видно из листингов 2.3 и 2.4, элементам массива автоматически назначены индексы, начиная с нулевого, как это принято в C-подобных языках программирования. Однако индекс начального элемента, а также порядок следования индексов можно изменять. Для этого в конструкции `array()` перед элементом указывается его индекс при помощи оператора `=>`. В листинге 2.5 первому элементу массива `$arr` назначается индекс 10, последующие элементы автоматически получают значения 11 и 12.

Листинг 3.5. Назначение индекса первому элементу массива

```
<?php
    $arr = array(10 => "Hello ", "world", echo "<pre>";
    print_r($arr);
```

```
    echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 3.5 будет следующий дамп массива \$arr:

```
Array
(
    [10] => Hello
    [11] => world
    [12] => !
)
```

В массиве \$arr из листинга 2.5 элементы с индексами 0-9 и 13 просто не существуют, обращение к ним воспринимается как обращение к неинициализированной переменной (т. е. переменной, имеющей значение NULL). Если в конфигурационном файле php.ini включено отображение замечаний (Notice), при попытке обращения к несуществующему элементу массива будет выдано соответствующее предупреждение.

Назначать индексы можно не только первому элементу, но и всем последующим элементам массива (листинг 2.6).

Листинг 3.6. Назначение индексов всем элементам массива

```
<?php
    $arr = array(10 => "Hello ", 9 => "world", 8 => "!");
    echo "<pre>";
    print_r($arr);
    echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 2.6 будет следующий дамп массива \$arr:

```
Array
(
    [10] => Hello
    [9] => world
    [8] => !
)
```

Непосредственное создание элементов

Еще одним способом создания массива является присвоение неинициализированным элементам массива новых значений (листинг 2.7).

Листинг 2.7. Создание массива при помощи квадратных скобок

```
<?php
    $arr[10] = "Hello
    $arr[11] = "world";
    $arr[12] = "!";
    echo "<pre>";
    print_r($arr);
    echo "</pre>";
?>
```

Скрипт из листинга 2.7 по результату выполнения эквивалентен листингу 2.5. Допускается и автоматическое назначение индексов элементам, для этого значение индекса просто не указывается в квадратных скобках (листинг 2.8).

Листинг 2.8. Автоматическое назначение индекса

```
<?php
  $arr[] = "Hello"
  $arr[] = "world";
  $arr[] = "!";
  echo "<pre>";
  print_r($arr) ; echo
  "</pre>";
?>
```

Создание массива: приведение типа

Еще один способ создания массива заключается в приведении скалярной переменной (т. е. переменной типа int, float, string или boolean) к типу array. Результатом этого становится массив, содержащий один элемент с индексом 0 и значением, равным значению переменной (листинг 2.9).

Листинг 2.9. Приведение переменной к массиву

```
<?php
  $var = "Hello world";
  $arr = (array) $var;
  echo "<pre>";

  print_r($arr);

  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 3.9 будет следующий дамп массива \$arr:

```
Array
(
  [0]=> Hello world
)
```

Использование специализированных функций

Помимо конструкции array(), существует еще несколько функций, которые позволяют создавать массивы. Их список приведен в табл. 2.1.

Таблица 2.1 – Функции создания массивов

Функция	Описание
array([...])	Создает массив из значений, переданных конструкции в качестве параметров
array_fill(\$start_index, \$num, \$value)	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

<code>explode(\$delimiter, \$str [, \$limit])</code>	Функция возвращает массив из строк, каждая из которых соответствует фрагменту исходной строки <code>\$str</code> , находящемуся между разделителями, определяемым аргументом <code>\$delimiter</code> . Необязательный параметр <code>\$limit</code> определяет максимальное количество элементов в массиве.
------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Если элементы массива должны содержать одинаковые значения, для его создания удобно воспользоваться функцией `array_fill()`, которая имеет следующий синтаксис:

```
array_fill($start_index, $num, $value)
```

Функция возвращает массив, содержащий `$num` элементов, имеющих значение `$value`. Нумерация индексов при этом начинается со значения `$start_index` (листинг 2.10).

Листинг 2.10. Создание массива при помощи функции `array_fill()`

```
<?php
// Создаем массив
$arr = array_fill(5, 6, "Hello world!");
// Выводим дамп массива
echo "<pre>";
print_r($arr);
echo "</pre>";
?>
```

Результатом работы скрипта из листинга 2.10 будут следующие строки:

```
Array
(
    [5] => Hello world!
    [6] => Hello world!
    [7] => Hello world!
    [8] => Hello world!
    [9] => Hello world!
    [10] => Hello world!
)
```

Еще одним удобным способом создания массивов является использование функции `range()`, которая позволяет создавать массивы для интервалов значений и имеет следующий синтаксис:

```
range($low, $high [, $step])
```

Параметр `$low` определяет начало интервала, а `$high` — его окончание. Необязательный параметр `$step` позволяет задать шаг, с которым будут создаваться элементы. В листинге 2.11 демонстрируется создание массива из 6 элементов, которые принимают значения в интервале от 0 до 5.

Листинг 2.11. Использование функции `range()`

```
<?php
$arr = range(0, 5);
echo "<pre>";
print_r($arr);
echo "</pre>";
?>
```

Результатом работы скрипта из листинга 2.11 будет следующий дамп массива:

```
Array
(
    [0] => 0
    [1] => 1
    [2] => 2
    [3] => 3
    [4] => 4
    [5] => 5
)
```

По умолчанию, массив заполняется значениями из интервала с шагом, равным единице, однако этот шаг можно изменить при помощи третьего параметра функции **range ()** (листинг 2.12).

В функции `range()` допускается использование отрицательного шага.

Листинг 2.12. Использование шага в функции `range ()`

```
<?php
    $arr = range(0, 1, 0.2);
    echo "<pre>";
    print_r($arr) ;
    echo "</pre>";
?>
```

Результатом работы скрипта из листинга 2.12 будет следующий дамп массива:

```
Array
(
    [0] => 0
    [1] => 0.2
    [2] => 0.4
    [3] => 0.6
    [4] => 0.8
    [5] => 1
)
```

Часто возникает задача получения массива за счет разбиения строки по какому-либо разделителю. В этом случае удобно воспользоваться функцией `explode()`, которая имеет следующий синтаксис:

```
explode($delimiter, $str [, $limit])
```

Функция возвращает массив из строк, каждая из которых соответствует фрагменту исходной строки `$str`, находящемуся между разделителями, определяемыми параметром `$delimiter`.

Необязательный параметр `$limit` определяет максимальное количество элементов в массиве. Оставшаяся (неразделенная) часть будет содержаться в последнем элементе. Пример использования функции `explode()` приведен в листинге 2.13.

Для того чтобы преобразовать массив в строку, используется обратная функция `implode()`.

Листинг 2.13. Использование функции `explode()`

```
<?php
$str = "Имя Фамилия e-mail";
$arr = explode(" ", $str);
echo "<pre>";
print_r ($arr);
echo "</pre>";
?>
```

Результатом работы скрипта из листинга 3.13 являются следующие строки:

```
Array
(
    [0] => Имя
    [1] => Фамилия
    [2] => e-mail
)
```

В листинге 2.13 строка `$str` разбивается на отдельные элементы массива `$arr` по пробельному символу. В качестве первого параметра функция `explode()` может использовать не одиночный символ, а целые последовательности, например, запятая и пробел " ," или перевод строки `"\r\n"`. Однако если необходимо разбить строку по сложному критерию, например, по пробелам, переводам строк, причем количество пробельных символов может быть произвольным, вместо функции `explode()` следует использовать регулярные выражения.

Многомерные массивы

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()`. В листинге 2.14 демонстрируется создание многомерного ассоциативного массива `$ship`.

Листинг 2.14. Создание многомерного массива

```
<?php
$ship = array(
    "Пассажирские корабли" => array("Лайнер", "Яхта", "Паром"),
    "Военные корабли" => array("Авианосец", "Линкор", "Эсминец"),
    "Грузовые корабли" => array("Сормовский", "Волго-Дон", "Окский")
);
echo "<pre>";
print_r($ship);
echo "</pre>";
?>
```

В результате такой инициализации будет создан массив следующей структуры:

```
Array
(
    [Пассажи́рские корабли] => Array
        (
            [0] => Лайнер
            [1] => Яхта
            [2] => Паром
        )
    [Военные корабли] => Array
        (
            [0] => Авианосец
            [1] => Линкор
            [2] => Эсминец
        )
    [Грузовые корабли] => Array
        (
            [0] => Сормовский
            [1] => Волго-Дон
            [2] => Окский
        )
)
```

В этом примере создан двумерный массив с размерами 3x3, т. е. получилось три массива, каждый из которых содержит в себе по три элемента. Следует отметить, что полученный массив является смешанным, т. е. в нем присутствуют как индексы, так и ключи ассоциативного массива — обращение к элементу `$hip['Пассажи́рские корабли'] [0]` возвратит значение "Лайнер".

Вывод массива на печать

С одним из способов вывода массива мы уже познакомились, это вывод дампа массива при помощи функции `print_r()` (листинг 3.15).

Листинг 2.15. Вывод дампа массива при помощи функции `print_r()`

```
<?php
$arr = array(1, 2, 3);
echo "<pre>";
print_r($arr);
echo "</pre>"; ■
?>
```

Это самый быстрый способ, который часто применяется для отладочных целей. Если элементы массива при выводе должны быть оформлены, то можно воспользоваться операторами цикла `for` или `while`. Для ассоциативных массивов предназначен специализированный оператор цикла `foreach`.

Итерационный цикл `for` имеет следующий синтаксис:

```
for(begin; condition; body)
{
```

```

    операторы;
}

```

Здесь оператор `begin` (инициализация цикла) — последовательность определений и выражений, разделяемая запятыми. Все выражения, входящие в инициализацию, вычисляются только один раз при входе в цикл. Как правило, здесь устанавливаются начальные значения счетчиков и параметров цикла. Смысл выражения-условия (`condition`) такой же, как и у циклов с пред- и постусловиями — цикл выполняется до тех пор, пока выражение `condition` истинно (`TRUE`), и прекращает свою работу, когда оно ложно (`FALSE`). При отсутствии выражения-условия предполагается, что его значение всегда истинно. Выражения `body` вычисляются в конце каждой итерации после выполнения тела цикла.

В листинге 2.16 демонстрируется обход индексного массива при помощи цикла `for`.

Листинг 2.16. Обход массива в цикле `for`

```

<?php
    $number = array("1", "2", "3");
    for($i=0; $i < count($number); $i++)
    {
        echo $number[$i];
    }
?>

```

Результатом работы скрипта из листинга 2.16 будет строка: 123.

Для подсчета количества элементов в массиве используется функция `count()` (см. разд. 3.3), которая принимает в качестве единственного параметра массив и возвращает количество элементов в нем. Так в листинге 2.16 для массива `$number` будет возвращено значение 3.

Обход массива в цикле можно организовать при помощи цикла `foreach`, который специально создан для ассоциативных массивов и имеет следующий синтаксис:

```

foreach($array as [$key =>] $value)
{
    операторы;
}

```

Цикл последовательно обходит элементы массива `$array` с первого до последнего, помещая на каждой итерации цикла ключ в переменную `$key`, а значение в переменную `$value`. Имена этих переменных могут быть любые (листинг 2.17).

В качестве первого аргумента конструкции `foreach` обязательно должен выступать массив, иначе конструкция выводит предупреждение.

Листинг 2.17. Обход массива в цикле `foreach`

```

<?php
    $number = array("first" => "1",
                   "second" => "2",

```

```

        "third" => "3");
    foreach($number as $index => $val) echo "$index = $val <br>";
?>

```

Результатом работы скрипта из листинга 2.17 будут следующие строки:

```

first = 1
second = 2
third = 3

```

Переменная `$index` для ключа массива необязательна и может быть опущена (листинг 2.18).

Листинг 2.18. Вариант обхода массива в цикле `foreach`

```

<?php
    $number = array("first" => "1",
                   "second" => "2",
                   "third" => "3");
    foreach($number as $val)
    {
        echo $val; // выведет 123
    }
?>

```

Обход многомерных массивов проводится с помощью вложенных циклов **foreach**, при этом число вложенных циклов соответствует размерности массива (листинг 3.19).

Листинг 2.19. Обход многомерных массивов в цикле

```

<?php
$ship = array(
    "Пассажирские корабли" => array("Лайнер", "Яхта", "Паром"),
    "Военные корабли" => array("Авианосец", "Линкор", "Эсминец"),
    "Грузовые корабли" => array("Сормовский", "Волго-Дон", "Окский")
);
foreach($ship as $key => $type)
{
    // вывод значений основных массивов
    echo "<b>$key</b><br>";
    foreach($type as $ship)
    {
        // вывод значений для каждого из массивов
        echo "<li>$ship</li><br>";
    }
}
?>

```

Результат выполнения скрипта из листинга 2.19 представлен на рис. 2.1.

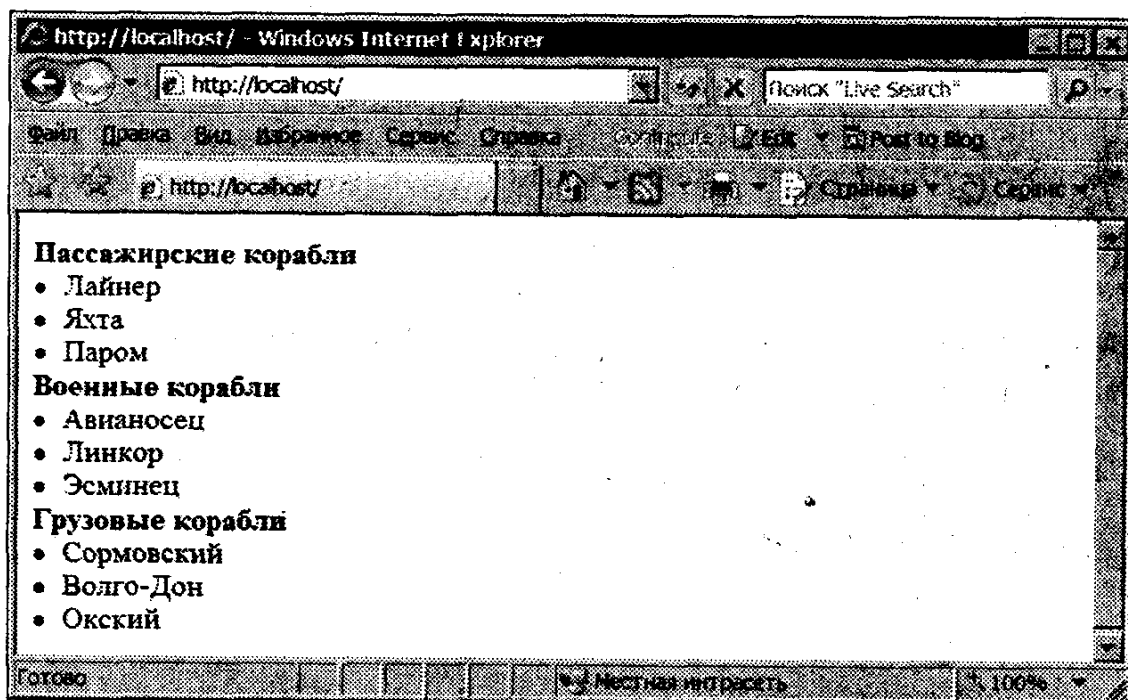


Рисунок 2.1 - Вывод многомерного массива

Количество элементов в массиве

Для манипуляций с массивами и их элементами часто необходимо определить количество элементов в массиве. В предыдущем разделе мы уже сталкивались с функцией `count()` для подсчета количества элементов в массиве для обхода их в цикле. Полный список функций, предназначенных для решения этой задачи, представлен в табл. 2.2.

Таблица 2.2 – Количество элементов в массиве

Функция	Описание
<code>count(\$array [, \$mode])</code>	Возвращает количество элементов массива <code>\$array</code> . Если <code>\$mode</code> принимает значение <code>COUNT_RECURSIVE</code> . Функция рекурсивно обходит многомерный массив, в противном случае подсчитывается количество элементов только на текущем уровне.
<code>sizeof()</code>	Синоним для функции <code>count()</code> .
<code>array_count_values(\$input)</code>	Подсчитывает количество уникальных значений среди элементов массива и возвращает ассоциативный массив, ключи которого – значения массива, а значения – количество их вхождений в массив <code>\$input</code> .

Для подсчета количества элементов в массиве предназначена функция `count()`, которая имеет следующий синтаксис:

```
count ($array [, $mode])
```

Возвращает количество элементов массива `$array`. В листинге 2.20 демонстрируется пример с использованием этой функции.

Листинг 2.20. Использование функции count()

```
<?php
    $car = array("жигули",
                "волга",
                "запорожец",
                "ока");
    echo count($car); // 4
?>
```

Если в качестве аргумента функции передается многомерный массив, то по умолчанию она возвращает размерность текущего измерения (листинг 2.21).

Листинг 2.21. Использование count() совместно с многомерными массивами

```
<?php
    $arr = array(
        array(1, 2, 3, 4),
        array(5, 6, 7, 8)
    );
    echo count($arr); // 2
    echo count($arr[0]); // 4
?>
```

В первом случае возвращается число 2, т. к. в первом измерении только 2 элемента это массивы `array(1, 2, 3, 4)` и `array(5, 6, 7, 8)`. Во втором случае в качестве аргумента передается уже массив `array(1, 2, 3, 4)`, в котором четыре элемента, о чем и сообщает функция `count()`.

Для того чтобы функция рекурсивно обходила многомерный массив, подсчитывая количество всех элементов (в том числе во вложенных массивах), необязательному параметру `$mode` необходимо передать в качестве значения константу `COUNT_RECURSIVE` (листинг 2.22).

Листинг 2.22. Подсчет элементов многомерного массива :

```
<?php
    $arr = array(
        array(1, 2, 3, 4),
        array(5, 6, 7, 8)
    );
    echo count($arr, COUNT_RECURSIVE); // 10
?>
```

Еще одной интересной функцией является `array_count_values()`, которая подсчитывает количество уникальных значений среди элементов массива и имеет следующий синтаксис:

```
array_count_values ($array)
```

В качестве результата возвращает ассоциативный массив, в качестве ключей которого выступают значения массива, а в качестве значения — количество их вхождений в массив `$array` (листинг 2.23).

Массив `$array` может иметь в качестве элементов лишь числа или строки, в противном случае генерируется предупреждение.

Листинг 2.23. Использование функции `array_count_values()`

```
<?php
$array = array (1, "hello", 1, "world", "hello");
$new = array_count_values ($array);

echo "<pre>";
print_r($new);
echo "</pre>";
?>
```

В результате выполнения скрипта из листинга 2.23 будет выведен следующий дамп массива `$new`:

```
Array
(
    [1] => 2
    [hello] => 2
    [world] => 1
)
```

Переменная или массив?

PHP является слаботипизированным языком программирования, при создании переменной или массива не обязательно указывать их тип. Причем за время работы программы, переменная может многократно изменять свой тип: становиться переменной или массивом. Особенно остро эта проблема встает, если функция, возвращающая массив (в случае неудачи), выдает логическое значение `FALSE`.

Если необходимо убедиться, является ли текущая переменная массивом, используют функцию `is_array()` (листинг 2.24).

Листинг 2.24. Использование функции `is_array()`

```
<?php
$arr = array(1, 2, 3);
if(is_array($arr)) echo "Это массив<br />";
else echo "Это не массив<br />";
if(is_array($arr[0])) echo "Это массив<br />";
else echo "Это не массив<br />";
?>
```

В качестве результата скрипт выводит следующие строки:

```
Это массив
Это не массив
```

6.5 Существует ли элемент массива?

Индексы массивов могут начинаться с чисел, отличных от нуля. В случае ассоциативных массивов ключи и вовсе могут быть произвольными. Проверить, существует тот или иной элемент массива, можно при помощи конструкции `isset ()` (листинг 2.25).

Листинг 2.25. Проверка существования элементов массива при помощи `isset()`

```
<?php
    $arr = array(5 => 1, 2, 3);
    for($i = 0; $i < 10; $i++)
    {
        if(isset($arr[$i]) ) echo "Элемент \$arr[$i] существует<br>";
        else echo "Элемент \$arr[$i] не существует<br>"•
    }
?>
```

В качестве результата скрипт из листинга 2.25 выводит следующие строки:

```
Элемент $arr[0] не существует
Элемент $arr[1] не существует
Элемент $arr[2] не существует
Элемент $arr[3] не существует
Элемент $arr[4] не существует
Элемент $arr[5] существует
Элемент $arr[6] существует
Элемент $arr[7] существует
Элемент $arr[8] не существует
Элемент $arr[9] не существует
```

Как получить список всех индексов массива?

Перебор индексов массива, как это демонстрировалось в предыдущем разделе, не всегда эффективен и возможен. Для массива со сложной структурой либо прибегают к специальному циклу `foreach` (см. *разд. 6.2*), либо получают список ключей массива при помощи функции `array_keys ()`, которая имеет следующий синтаксис:

```
array_keys($arr [, $search_value [, $strict]])
```

Функция возвращает массив ключей для массива `$arr`. Если указан необязательный параметр `$search_value`, возвращаются только те ключи, которые указывают на элементы со значением `$search_value`. Если необязательный параметр `$strict` принимает значение **TRUE**, то при сравнении значения элемента массива с `$search value` будет использоваться оператор эквивалентности `===`, в противном случае будет использован оператор равенства `==`. В листинге 2.26 приводится пример использования функции.

Листинг 2.26. Использование функции `array_keys()`

```
<?php
    // Исходный массив
    $arr = array(0 => 100, "color" => "red");
```

```
// Получаем массив ключей

$key = array_keys($arr);
// Выводим дамп массива
echo "<pre>";
print_r($key) ;
echo "</pre>";
?>
```

Результат:

```
Array
(
    [0] => 0
    [1] => color
)
```

В листинге 2.27 демонстрируется использование параметра `$search_value`. Исходный массив содержит три элемента со значением "blue", при помощи функции `array_keys()` извлекаются их ключи.

Листинг 2.27. Извлечение ключей элементов со значением "blue"

```
<?php
// Исходный массив
$arr = array("blue", "red", "green", "blue", "blue");
// Получаем массив ключей
$key = array_keys($arr, "blue");
// Выводим дамп массива
echo "<pre>";
print_r($key) ;
echo "</pre>";
?>
```

Результат:

```
Array
(
    [0] => 0
    [1] => 3
    [2] => 4
)
```

Содержит ли массив заданный элемент?

Конструкция `isset()` выполняет проверку существования элемента с заданным ключом, однако помимо этого зачастую требуется выяснить, входит ли в состав того или иного массива элемент с заданным значением.

Функция `in_array()` осуществляет поиск значения в массиве и имеет следующий синтаксис:

```
in_array($value, $arr [, $strict])
```

Функция проверяет, существует ли значение `$value` в массиве `$arr`, и возвращает `true`, если значение найдено, и `false` в противном случае. Если необязательный параметр `$strict` принимает значение `true`, то при сравнении

значения элемента массива с `$search_value` будет использоваться оператор эквивалентности `===` (т. е. сравнению будет подвергаться не только значение, но и тип элемента), в противном случае будет использован оператор равенства (листинг 228).

Листинг 2.28. Поиск элемента в массиве .

```
<?php
    $number = array(0.57, '21.5', 40.52);
    if (in_array(21.5, $number)) echo "Значение 21.5 найдено";
    else echo "Ничего не найдено";
?>
```

В результате будет выведена фраза:

Значение 21.5 найдено

Заметим, что найденный элемент массива `'21.5'` взят в одинарные кавычки, т. е. относится к строковому типу, в отличие от других элементов массива `$number`. В приведенном варианте использования функции это различие не фиксируется. Для того чтобы функция использовала для сравнения оператор эквивалентности `===`, вместо оператора равенства `==` необходимо третий необязательный параметр `$strict` установить в значение `TRUE` (Листинг 2.29)

Листинг 2.29. Поиск элемента в массиве с различием по типу

```
<?php
    $number = array(0.57, '21.5', 40.52);
    if (in_array(21.5, $number, true)) echo "Значение 21.5 найдено";
    else echo "Ничего не найдено";
?>
```

В результате будет выведена фраза:

Ничего не найдено

В этом случае ничего найдено не будет, т. к. тип элемента, передаваемого функции `in_array()` (`float`), отличается от типа элемента в массиве (`string`). При таком вызове функции элемент `21.5` будет найден только, если он так же будет взят в кавычки (т. е. тоже будет строкового типа):

```
if (in_array('21.5', $number, true))
```

Поиск ключа по значению

По значению ключа или индекса очень просто получить значение искомого элемента: достаточно поместить ключ в квадратные скобки. Однако не менее редко может возникать обратная задача — поиск ключа ассоциативного массива по значению. Для проверки существования ключа в

массиве и поиска ключа по значению элемента массива предназначены специализированные функции из табл. 2.3.

Таблица 2.3. Проверка существования ключей

Функция	Описание
<code>array_key_exists(\$key, \$arr)</code>	Возвращает true, если в массиве \$arr присутствует ключ \$key, в противном случае возвращается false
<code>array_search(\$value, \$arr [, \$strict])</code>	Ищет значение \$value в массиве \$arr и, если значение найдено, возвращает соответствующий ключ, в противном случае возвращается false. Если необязательный параметр \$strict принимает значение true, то при сравнении значения элемента массива с \$value будет использоваться оператор эквивалентности ===, в противном случае будет использован оператор равенства ==

Для поиска заданного ключа в массиве можно воспользоваться функцией `array_key_exists()`, которая имеет следующий синтаксис:

```
array_key_exists($key, $arr)
```

Функция возвращает true, если ключ \$key найден в массиве \$arr (листинг 2.30).

Листинг 2.30. Поиск ключей массива

```
<?php
    $arr = arr("first_num" => 1, "second_num" => 2);
    if (array_key_exists("first_num", $arr) echo "OK";
?>
```

Найти ключ массива по значению позволяет функция `array_search()`, которая имеет следующий синтаксис:

```
array_search($value, $arr [, $strict])
```

Функция ищет значение \$value в массиве \$arr и, если значение найдено, возвращает соответствующий ключ, в противном случае возвращается false. Если необязательный параметр \$strict принимает значение true, то при сравнении значения элемента массива с \$value будет использоваться оператор эквивалентности ===, в противном случае будет использован оператор равенства ==. Пример использования функции приводится в листинге 2.31.

Листинг 2.31. Использование функции `array_search()`

```
<?php
    $array = array(0 => 'blue', 1 => 'red', 2 => 'green', 3 => 'red');
    $key = array_search('green', $array); // $key = 2;
    $key = array_search('red', $array); // $key = 1;
?>
```

6.9 Сумма элементов массива

Функция `array_sum()` возвращает сумму всех числовых элементов массива и имеет следующий синтаксис:

```
array_sum($arr)
```

Тип возвращаемого числа определяется типом значений элементов массива. Пример с использованием функции `array_sum()` демонстрируется в листинге 2.32.

Листинг 2.32. Использование функции `array_sum()`

```
<?php
    $arr = array(1,2,3,4,5);
    $sum = array_sum($arr);
    echo $sum; // выводит 15
?>
```

Случайные элементы массива

Для получения случайного значения индексного массива можно использовать функцию `rand()`, которая возвращает случайное число из заданного диапазона. Функция `rand()` имеет следующий синтаксис:

```
rand([$min, $max]);
```

Необязательные параметры `$min` и `$max` задают соответственно начало и конец интервала, из которого выбирается случайное значение. При помощи функции **`rand()`** можно извлечь случайный индекс массива, указав в качестве начала интервала 0, а в качестве конца количество элементов в массиве за вычетом единицы (листинг 2.33).

Листинг 2.33. Получение случайного элемента массива

```
<?php
    // Объявляем массив
    $arr = array(0, 1, 2, 3, 4, 5, 6, 7, 8, 9);
    // Получаем случайный индекс массива
    $index = rand(0, count($arr) - 1);
    // Выводим случайный элемент массива
    echo $arr[$index];
?>
```

Подход, описанный в листинге 2.25, применим только для индексных массивов, индексы которых начинаются с 0 и не имеют перерывов в нумерации. Для всех остальных массивов используется функция `array_rand()` (табл. 2.4), которая возвращает массив выбранных случайным образом индексов элементов массива `$arr`. Функция имеет следующий синтаксис:

```
array_rand ($arr [, $num_req])
```

Функция выбирает случайное значение из массива `$arr`. Если параметр `$num_req` не указывается, возвращается ключ для случайного элемента массива `$arr`, если в параметре `$num_req` задается число больше 1, возвращается массив со случайным набором ключей массива `$arr` (листинг 2.34).

Таблица 2.4. Случайные элементы массива

Функция	Описание
---------	----------

<code>array_rand (\$arr [, \$num req])</code>	Выбирает одно или несколько случайных значений из массива. Если параметр <code>\$num_req</code> не указывается, возвращается ключ для случайного элемента массива <code>\$arr</code> , если в параметре <code>\$num_req</code> задается число больше 1, возвращается массив со случайным набором
<code>shuffle (\$arr)</code>	Перемешивает элементы массива <code>\$arr</code> в случайном порядке

Листинг 2.34. Использование функции `array_rand()`

```
<?php
// Иницируем массив
$arr = array("синий", "желтый", "зеленый", "красный", "оранжевый");
$rand_keys = array_rand($arr, 2);
echo "<pre>";
print_r($rand_keys);
echo "</pre>";
?>
```

Еще одной полезной функцией для получения случайных значений является функция `shuffle()` (табл. 2.4), которая перемешивает элементы массива случайным образом и имеет следующий синтаксис:

```
shuffle($array)
```

Листинг 2.35. Функция `shuffle ()`

```
<?php
$arr = array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
shuffle($arr);
echo "<pre>";
print_r($arr);
echo "</pre>";
?>
```

Результат работы функции `shuffle()` может выглядеть следующим образом:

```
Array
(
    [0] => 7
    [1] => 9
    [2] => 8
    [3] => 1
    [4] => 3
    [5] => 10
    [6] => 2
    [7] => 4
    [8] => 5
    [9] => 6
)
```

Слияние массивов

Оператор плюс `+`, позволяющий складывать значения двух переменных, применим и для массивов (листинг 2.36).

Наряду с оператором плюс +, допускается использование оператора +=.

Листинг 2.36. Слияние массивов при помощи оператора +

```
<?php
  $fst = array(1 => "one", 2 => "two");
  $snd = array(3 => "three", 4 => "four");
  $sum = $fst + $snd;
  echo "<pre>";
  print_r($sum);
  echo "</pre>";
?>
```

В результате выполнения скрипта из листинга 2.36 в окно браузера будет выведен следующий дамп нового массива \$sum:

```
Array
(
    [1] => one
    [2] => two
    [3] => three
    [4] => four
)
```

Если в обоих складываемых массивах присутствуют элементы с одинаковыми индексами, в результирующий массив попадает только один элемент из правого массива (листинг 2.37).

Листинг 2.37. В результирующий массив попадает только один элемент с одинаковыми индексами

```
<?php
  $fst = array("one", "two");
  $snd = array("three", "four", "five");
  $sum = $fst + $snd;
  echo "<pre>";
  print_r ($sum) ;
  echo "</pre>";
?>
```

В результате выполнения скрипта из листинга 2.37 в окно браузера будет выведен следующий дамп нового массива \$sum:

```
Array
(
    [0] => one
    [1] => two
    [2] => five
)
```

Для того чтобы в результирующий массив попали элементы обоих массивов, вместо оператора + используют специальные функции (табл. 3.5).

Таблица 3.5. Функции слияния массивов

Функция	Описание
---------	----------

<code>array_merge (\$array1 [, \$array2 [, ...]])</code>	Сливает массивы <code>\$array1, \$array2, ...</code> в один и возвращает его в качестве значения
<code>array_merge_recursive (\$array1 [, \$array2, [, ...]])</code>	Рекурсивно сливает массивы <code>\$array1, \$array2, ...</code> в один и возвращает его в качестве значения

В листинге 2.38 приводится пример использования функции `array_merge()`.

Функция `array_merge()` может принимать в качестве параметров более чем два массива.

Листинг 2.38. Использование функции `array_merge()`

```
<?php
    $fst = array("one", "two");
    $snd = array("three", "four", "five");
    $sum = array_merge($fst, $snd);
    echo "<pre>";
    print_r($sum);
    echo "</pre>";
?>
```

В результате выполнения скрипта в окно браузера будет выведен следующий дамп нового массива `$sum`:

```
Array
(
    [0] => one
    [1] => two
    [2] => three
    [3] => four
    [4] => five
)
```

В арсенале PHP имеется функция `array_merge_recursive()`, которая ведет себя аналогично `array_merge()` в отношении индексных массивов и как оператор `+` в отношении ассоциативных массивов (листинг 2.39).

Листинг 2.39. Использование функции `array_merge_recursive`

```
<?php
    $ar1 = array("color" => array ("favorite" => "red"), 5);
    $ar2 = array(10, "color" => array ("favorite" => "green", "blue"));
    $sum = array_merge_recursive ($ar1, $ar2);
    echo "<pre>";
    print_r($sum);
    echo "</pre>";
?>
```

В результате выполнения скрипта из листинга 2.39 в окно браузера будет выведен следующий дамп нового массива `$sum`:

```
Array
(
    [color] => Array
        (
            [favorite] => Array
```

```

        (
            [0] => red ,
            [1] => green
        )
        [0] => blue
    )
    [0] => 5
    [0] => 10
)

```

Преобразование каждого элемента массива

В общем случае для того, чтобы обработать элементы массива, достаточно обойти их в цикле и применить требуемые операции к каждому из элементов. Так как эта задача возникает достаточно часто, в PHP предусмотрена специализированная функция `array_walk()`, которая имеет следующий синтаксис:

```
array_walk ($arr, $funcname [, $userdata])
```

Функция применяет пользовательскую функцию `$funcname` ко всем элементам массива `$arr`. Параметр `$userdata` используется в качестве дополнительного параметра пользовательской функции.

В пользовательскую функцию передаются два или три аргумента: значение текущего элемента, его индекс и аргумент `$userdata`. Последний аргумент является необязательным. В случае если `$funcname` требует более трех аргументов, при каждом ее вызове будет выдаваться предупреждение, и, чтобы они не выдавались, нужно поставить знак `@` перед функцией `array_walk()`.

Пусть необходимо вывести все элементы массива. Для этого можно сначала написать функцию, которая будет их выводить, а затем вызвать ее для каждого из элементов массива (листинг 2.40).

Функция `array_walk()` имеет модификацию `array_walk_recursive()`, которая позволяет применять пользовательскую функцию к многомерным массивам.

Листинг 2.40. Использование функции `array_walk()`

```

<?php
$name = array ("m"=>"maksim", "i"=>"igor", "s"=>"sergey");
function print_array ($item, $key)
{
    echo "$key=>$item<br>\n";
}
/* теперь применим функцию print_array '
к каждому элементу массива $name */
array_walk ($name, 'print_array');
?>

```

Результат:

```
m => maksim
```

```
i => igor
s => sergey
```

Функция `array_map()` также предназначена для применения пользовательской функции к элементам массивов. Функция имеет следующий синтаксис:

```
array_map ($callback, $arr [, ...])
```

Функция применяет пользовательскую функцию `$callback` ко всем элементам массивов, указанных в последующих параметрах. Количество параметров, передаваемых функции обратного вызова, должно совпадать с количеством массивов, переданным функции `array_map()`. В листинге 2.41 демонстрируется работа функции — массив `$arr` преобразуется в массив `$sub`, элементы которого представляют кубическую степень от элементов массива `$arr`.

Листинг 2.41. Возведение элементов массива в куб

```
<?php
// Пользовательская функция для возведения
// элементов массива в куб
function cube($n)
{
    return $n*$n*$n;
}
// Исходный массив
$arr = array(1, 2, 3, 4, 5);

// Возводим элементы массива в куб
$sub = array_map("cube", $arr);

// Выводим дамп массива
echo "<pre>";
print_r($sub);
echo "</pre>";
?>
```

Результат:

```
Array
(
    [0] => 1
    [1] => 8
    [2] => 27
    [3] => 64
    [4] => 125
)
```

Получение уникальных элементов массива

Массивы могут содержать повторяющиеся элементы, часто требуется получить копию массива, содержащую только уникальные значения. Для

этого удобно воспользоваться функцией `array_unique()`, которая имеет следующий синтаксис:

```
array_unique($array)
```

В листинге 2.42 приводится пример использования функции.

Листинг 2.42. Получение массива с уникальными значениями

```
<?php
$input = array("a" => "green", "red", "b" => "green", "blue", "red");
$result = array_unique($input) ;
echo "<pre>";
print_r($result);
echo "</pre>";
?>
```

Результатом работы скрипта будет следующий дамп массива `$result`:

```
Array
(
    [a] => green
    [0] => red
    [1] => blue
)
```

Преобразование элементов массива в переменные

Массивы, как и обычные переменные, могут выступать в качестве аргументов и возвращаемых значений пользовательских функций (листинг 2.43).

Листинг 2.43. Функция, возвращающая массив

```
<?php
// Функция, возвращающая массив с годом, месяцем и днем
function get_date($time)
{
    return array(date("Y", $time), date("m", $time), date("d", $time));
}
// Получаем текущую дату
$arr = get_date(time());
// Выводим дамп массива $arr
echo "<pre>";
print_r($arr);
echo "</pre>";
?>
```

В качестве аргумента функция `getdate()` принимает время в формате UNIXSTAMP, которое при помощи функции `date()` преобразует в три числа (год, месяц и день) составляющие элементы результирующего массива. В качестве результата, скрипт из листинга 2.43 выводит следующий дамп массива `$arr`:

```
Array
(
    => 2010
```

```
=> 05
=> 12
)
```

Получение результата выполнения функции в виде массива не всегда удобно, особенно, если количество элементов в массиве не велико, или не все они требуются для дальнейшей работы. В PHP для преобразования элементов массива в переменные предназначена специальная конструкция `list()`. В листинге 2.44 демонстрируется преобразование результирующего массива функции `get_date()` в три переменные `$year`, `$month` и `$day`, соответствующие году, месяцу и дню.

Листинг 2.44. Использование конструкции `list()`

```
<?php
// Функция, возвращающая массив с годом, месяцем и днем
function get_date($time)
{
    return array (date ("Y", $time) , date("m", $time), date("d", $time) ) ;
}
// Получаем текущую дату
list($year, $month, $day) = get_date(time());
echo "Год — $year<br />"; // Год — 2012
echo "Месяц — $month<br />"; // Месяц — 02
echo "День — $day<br />"; // День — 18
?>
```

Следует отметить, что конструкция `list()` работает только с числовыми массивами, нумерация индексов которых начинается с нуля. В листинге 2.45 приводится альтернативная реализация функции `get_date()`, которая возвращает ассоциативный массив. Однако попытка использования конструкции `list()` для заполнения переменных `$year`, `$month` и `$day` приводит к выдаче замечания о том, что переменные не были инициализированы.

Листинг 2.45. Конструкция `list()` не работает с ассоциативными массивами

```
<?php
// Функция, возвращающая массив с годом, месяцем и днем
function get_date($time)
{
    return array("year" => date("Y", $time),
                "month" => date("m", $time),
                "day" => date("d", $time) ) ;
}

// Получаем текущую дату
list($year, $month, $day) = get_date(time());
echo "Год — $year<br />"; // Notice: Undefined offset
echo "Месяц — $month<br />"; // Notice: Undefined offset
echo "День — $day<br />"; // Notice: Undefined offset
?>
```

Еще одной интересной особенностью конструкции `list()` является тот факт, что количество элементов в разбираемом массиве и в круглых скобках

конструкции **list()** может не совпадать. Если элементов в массиве больше, чем нужно, лишние просто будут отброшены, если меньше — часть переменных останется неинициализированной. Более того, часть первых элементов массива может быть пропущена, для этого достаточно указать соответствующее количество запятых (листинг 2.46).

Листинг 2.46. Получаем лишь месяц

```
<?php
// Функция, возвращающая массив с годом, месяцем и днем
function get_date($time)
{
    return array (date ("Y", $time), date("m", $time), date("d", $time));
}
// Получаем лишь месяц
list(, $month) = get_date(time());
echo "Месяц — $month<br // Месяц — 02
?>
```

Сортировка массивов

PHP предоставляет разработчику широкие возможности для сортировки массивов (табл. 2.6).

Таблица 2.6. Функции для сортировки массивов

Функция	Описание
sort (\$array [, \$sort flags])	Сортирует массив \$array в прямом порядке. Параметр \$sort_flags задает параметры сортировки: SORT_REGULAR – обычная сортировка; SORT_NUMERIC – сравнивать элементы как числа; SORT_STRING – сравнивать элементы как строки; SORT_LOCALE_STRING – сравнивать элементы как строки, основываясь на текущей локали
rsort (\$array [, \$sort flags])	Сортирует массив \$array в обратном порядке
asort (\$array [, \$sort flags])	Сортирует массив \$array в прямом порядке с сохранением отношения ключ-значение
arsort (\$array [, \$sort flags])	Сортирует массив \$array в обратном порядке с сохранением отношения ключ-значение
natsort (\$array)	Осуществляет "естественную" сортировку массива \$array, учитывая регистр
natcasesort (\$array)	Осуществляет "естественную" сортировку массива \$array, без учета регистра
krsort (\$array [, \$sort_flags])	Сортирует массив \$array в обратном порядке по ключам с сохранением отношения ключ-значение
ksort (\$array [, \$sort_flags])	Сортирует массив \$array в прямом порядке по ключам с сохранением отношения ключ-значение
usort (\$array, \$cmp_function)	Сортирует массив \$array с использованием пользовательской функции сравнения элементов \$cmp_function

<code>uasort (\$array, \$cmp function)</code>	Сортирует массив \$array с использованием пользовательской функции сравнения элементов \$cmp_function и сохранением отношения ключ-
<code>uksort (\$array, \$cmp function)</code>	Сортирует массив \$array по ключам с использованием пользовательской функции сравнения элементов \$cmp_function и сохранением отношения ключ-значение
<code>array_multisort (\$ar1 [, \$arg [, ... [, ...]])</code>	Сортирует один многомерный массив или подвергает сортировке несколько массивов сразу

Функция `sort()` позволяет сортировать массив `$arr` по возрастанию и имеет следующий синтаксис:

```
sort($arr [, $sort_flags])
```

Необязательный аргумент `$sort_flags` указывает, как именно должны сортироваться элементы (задает флаги сортировки). Допустимы следующие значения этого аргумента:

- `SORT_REGULAR` – задает нормальное сравнение элементов;
- `SORT_NUMERIC` – сравнивает элементы как числа;
- `SORT_STRING` – сравнивает элементы как строки;
- `SORT_LOCALE_STRING` – сравнивает элементы как строки, основываясь на текущей локали.

В листинге 2.47 приводится пример использования функции `sort()`.

Листинг 2.47. Сортировка массива по возрастанию

```
<?php
    $number = array("2", "1", "4", "3", "5"); // исходный массив
    echo "до сортировки: <br>";
    for($i=0; $i < count($number); $i++)
    {
        echo "$number[$i]
    }
    sort($number); // сортируем массив по возрастанию
    echo "<br> после сортировки: <br>";
    for($i = 0; $i < count($number); $i++)
    {
        echo "$number[$i]
    }
?>
```

Результат:

```
до сортировки:
2 1 4 3 5
после сортировки:
1 2 3 4 5
```

Сортировка строк происходит по старшинству первой буквы в алфавите. Такую сортировку часто называют сортировкой в альфа-бета-порядке. К примеру, если имеется массив:

```
array("one",
      "two",
      "abs",
      "three",
      "uic",
      "for",
      "five")
```

то применение к нему функции `sort` приведет к следующему результату:

```
[0] => abs
[1] => five
[2] => for
[3] => one
[4] => three
[5] => two
[6] => uic
```

Функция `rsort()` сортировки массива по убыванию имеет следующий синтаксис:

```
rsort ($arr [, $sort_flags])
```

Во всем остальном ведет себя аналогично функции `sort()`. В листинге 2.48 приводится пример использования функции `rsort ()`.

Листинг 2.48. Сортировка массива по убыванию

```
<?php
  $number = array("2", "1", "4", "3", "5"); // исходный массив
  echo("до сортировки: <br>");
  for($i=0; $i < count($number); $i++)
  {
    echo "$number[$i]
  }
  rsort($number); // сортируем массив по убыванию
  echo "<br> после сортировки: <br>";
  for($i=0; $i < count($number); $i++)
  {
    echo "$number[$i]
  }
?>
```

Результат:

```
до сортировки:
2 1 4 3 5
после сортировки:
5 4 3 2 1
```

Функция `asort()` осуществляет сортировку ассоциативного массива по возрастанию и имеет следующий синтаксис:

```
asort ($arr [, $sort_flags])
```


Функция `asort()` сортирует массив `$arr` так, чтобы его значения шли в алфавитном (если это строки) или возрастающем (для чисел) порядке. Значения необязательного аргумента `$sort_flags` такие же, как и для функции `sort()`. Важное отличие этой функции от функции `sort()` состоит в том, что при применении функции `asort()` сохраняются связи между ключами и соответствующими им значениями (листинг 2.49), чего нет в функции `sort()` (там эти связи попросту разрываются).

Листинг 2.49. Сортировка функцией `asort()`

```
<?php
    $arr = array("a" => "one",
                "b" => "two",
                "c" => "three",
                "d" => "four");
    asort($arr);
    foreach($arr as $key => $val)
    {
        echo " $key => $val ";
    }
?>
```

Результат:

```
d => four a => one c => three b => two
```

Как видно, связи "ключ-значение" сохранились при сортировке. При сортировке массива при помощи функции `sort()`, результат бы выглядел следующим образом:

```
d => four 1 => one 2 => three 3 => two
```

Функция `arsort()` аналогична функции `asort()`, только она упорядочивает массив не по возрастанию, а по убыванию.

```
arsort($arr [, $sort_flags])
```

Функция `ksort()` осуществляет сортировку не по значениям, а по ключам массива в порядке их возрастания, при этом связи между ключами и значениями сохраняются. Функция имеет следующий синтаксис:

```
ksort ($arr [, $sort_flags])
```

Значения аргумента `$sort_flags` такие же, как и функции `sort()`. В листинге 2.50 приводится пример использования функции `ksort()`.

Листинг 2.50. Сортировка по индексам массива

```
<?php
    $arr = array("a" => "one",
                "d" => "four",
                "c" => "three",
                "b" => "two"
                );
    ksort($arr);
    foreach($arr as $key => $val)
    {
```

```

        echo (" $key => $val ");
    }
?>

```

Результат:

```
a => one b => two c => three d => four
```

Функция `krsort()` осуществляет сортировку элементов массива по ключам в обратном порядке (по убыванию). Во всем остальном аналогична функции `ksort()`. Функция имеет следующий синтаксис:

```
krsort ($arr [, $sort_flags])
```

Функция **`natsort()`** выполняет "естественную" сортировку массива и имеет следующий синтаксис:

```
natsort($arr)
```

Под естественной сортировкой понимается сортировка элементов таким образом, как их отсортировал бы человек. Приведем пример. Пусть у нас есть несколько файлов с именами

```
file1.txt
file10.txt
file2.txt
file12.txt
file20.txt
```

При обычной сортировке файлы будут расположены в следующем ("машинном") порядке:

```
file1.txt
file10.txt
file12.txt
file2.txt
file20.txt
```

Естественная же сортировка приведет к следующему результату:

```
file1.txt
file2.txt
file10.txt
file12.txt
file20.txt
```

Производится сортировка с помощью функции `natsort()`, принимающей в качестве параметра сортируемый массив (листинг 2.51).

Листинг 2.51. Естественная сортировка

```

<?php
    $arr_first = $arr_second =
        array("file12.txt",
            "file10.txt",
            "file2.txt",
            "file1.txt");
    sort($arr_first);
    echo "Обычная сортировка<br>";

```

```

echo "<pre>";
print_r($arr_first);
echo "</pre>";

natsort($arr_second);
echo "<br>Естественная сортировка<br>";
echo "<pre>";
print_r($arr_second);
echo "</pre>";
?>

```

Результат:

Обычная сортировка

Array

```

(
  [0] => file1.txt
  [1] => file10.txt
  [2] => file12.txt
  [3] => file2.txt
)

```

Естественная сортировка

Array

```

(
  [3] => file1.txt
  [2] => file2.txt
  [1] => file10.txt
  [0] => file12.txt
)

```

Функция `array_multisort()` может быть использована для сортировки сразу нескольких массивов или одного многомерного массива. Функция имеет следующий синтаксис:

```
array_multisort($ar1 [, $arg [, ... [, ...]])
```

Массивы, которые передаются функции `array_multisort()`, должны содержать одинаковое количество аргументов и все вместе воспринимаются как своеобразная таблица. Сортировке подвергается лишь первый массив, а значения последующих массивов выстраиваются в соответствии с ним (рис. 2.2). В листинге 2.52 приводится пример сортировки двух массивов `$arr1` и `$arr2`.

array_multisort(arr1, \$arr2, ..., \$arrN)

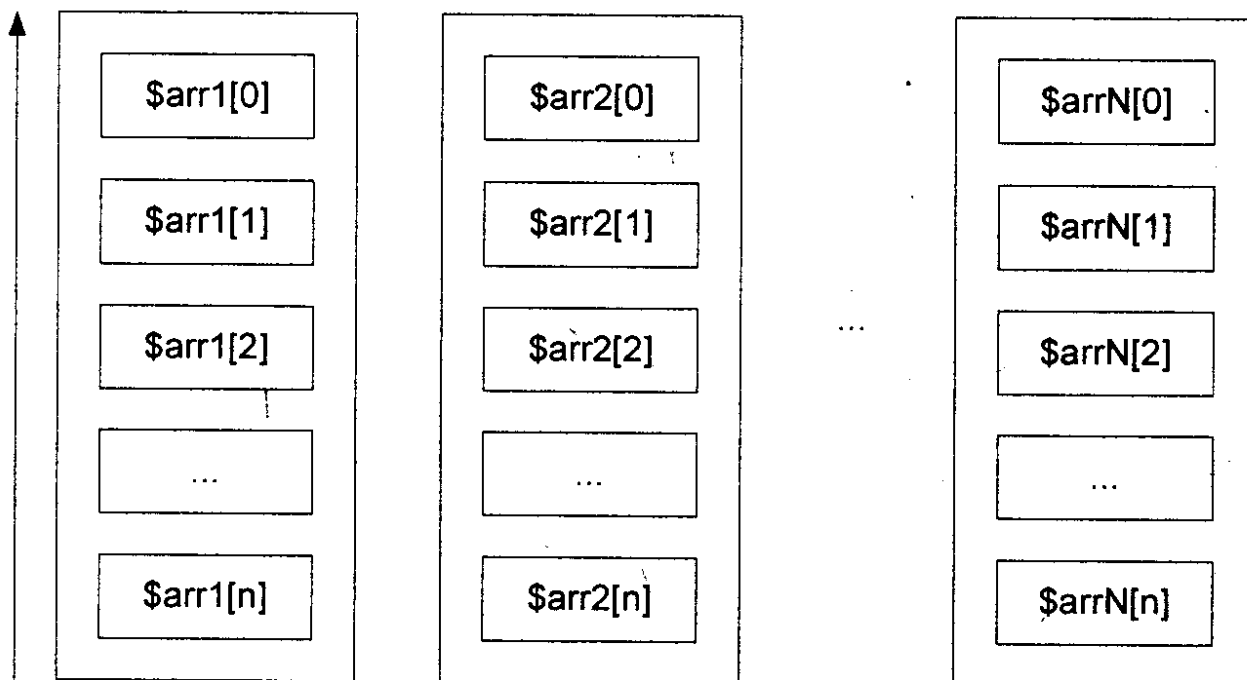


Рисунок 2.2 – Аргументы функции **array_multisort ()** образуют таблицу

Листинг 2.52. Использование функции array_multisort()

```
<?php
    $arr1 = array(3, 4, 2, 7, 1, 5);
    $arr2 = array("world", "Hello", "yes", "no", "apple", "wet");
    array_multisort($arr1, $arr2);
    echo "<pre>";
    print_r($arr1);
    print_r($arr2);
    echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 2.52 будут следующие дампы массивов:

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 4
    [4] => 5
    [5] => 7
)
Array
(
    [0] => apple
    [1] => yes
```

```

[2] => world
[3] => Hello
[4] => wet
[5] => no
)

```

Как видно из результатов, первый массив был отсортирован по возрастанию, а элементы второго массива выстроены в соответствии с первым так, как если бы между элементами двух массивов существовала связь.

Помимо массивов функция `array_multisort()` в качестве аргументов может принимать константы, задающие порядок сортировки {табл. 2.7}.

Группы констант `SORT_ASC` и `SORT_DESC`, а также `SORT_REGULAR`, `SORT_NUMERIC` и `SORT_STRING` являются взаимоисключающими.

Таблица 2.7. Константы, определяющие поведение функции `array_multisort()`

Функция	Описание
<code>SORT_ASC</code>	Сортировать в возрастающем порядке
<code>SORT_DESC</code>	Сортировать в убывающем порядке
<code>SORT_REGULAR</code>	Сортировать элементы обычным способом
<code>SORT_NUMERIC</code>	Сортировать элементы в предположении, что это числа
<code>SORT_STRING</code>	Сортировать элементы в предположении, что это строки

В листинге 2.53 приводится модифицированный вариант скрипта, сортирующий таблицу в обратном порядке.

Листинг 3.53. Использование констант

```

<?php
$arr1 = array(3, 4, 2, 7, 1, 5);
$arr2 = array("world", "Hello", "yes", "no", "apple", "wet");
array_multisort($arr1, SORT_DESC, SORT_NUMERIC, $arr2);
echo "<pre>";
print_r($arr1);
print_r($arr2);
echo "</pre>";
?>

```

Вывод иерархических данных

Для организации сложных структур не обязательно создавать массивы глубокой степени вложенности. Сложные иерархические данные (такие как сообщения на форуме или в комментариях в виде лестницы), меню сайта можно создать в рамках двумерного массива, предусмотрев отдельный элемент для указания предка. На рис. 2.3 представлен типичная лестничная структура, где ответ под сообщением немного сдвинут вправо.

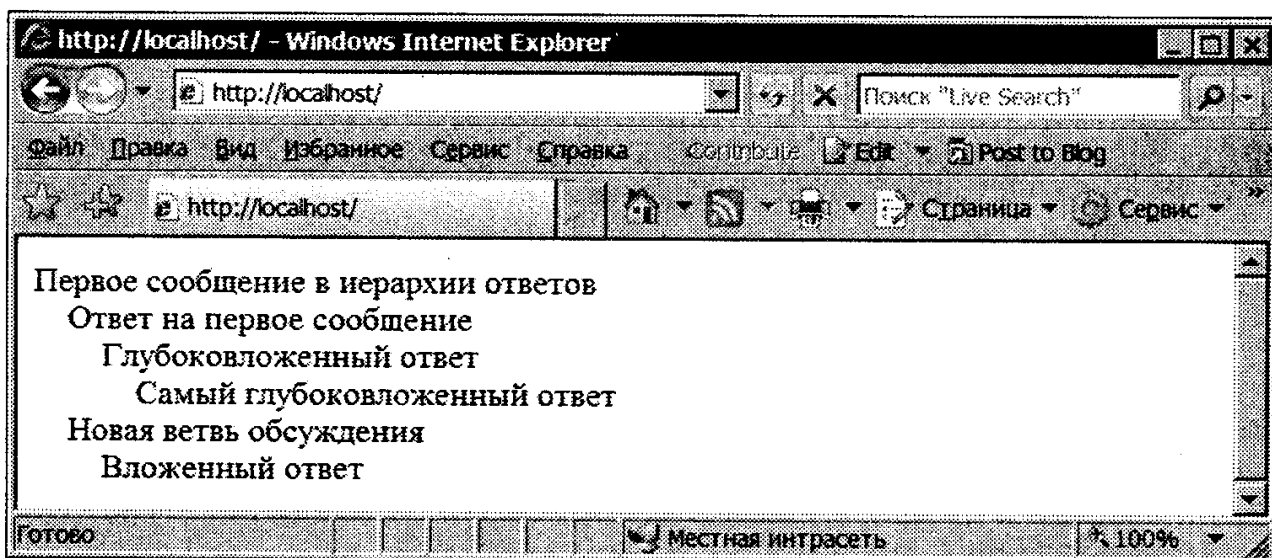


Рисунок 2.3 – Лестничная структура ответов

Для формирования такой структуры достаточно двумерного массива `$comments`, представленного в листинге 2.54. Каждый элемент такого массива, помимо текста сообщения в элементе с ключом "text", в дополнительном элементе с ключом "id parent" содержит ссылку на индекс родительского сообщения.

Листинг 2.54. Массив для лестничной структуры

```
<?php
$coiranents = array (
  1 => array(
    "text" => "Первое сообщение в иерархии ответов",
    "id_parent" => 0),
  2 => array(
"text" => "Ответ на первое сообщение",
  "id_parent" => 1),
  3 => array(
  "text" => "Глубоковложенный ответ",
  "id_parent" =>2),
  4 => array(
  "text" => "Новая ветвь обсуждения",
  "id_parent" => 1),
  5 => array(
  "text" => "Вложенный ответ",
  "id_parent" => 4),
  6 => array(
  "text" => "Самый глубоковложенный ответ",
  "id_parent" =>3)
);
?>
```

Для удобства оперирования элементами из массива `$comments`, как правило, создают дополнительный двумерный массив `$parents`, который соотносит индексы родительских сообщений и ответов на них (листинг 2.55).

Листинг 2.55. Вспомогательный массив `$parents`

```

<?php
// Формируем массив родителей и их потомков
$parents = array();
foreach($comments as $id => $elements)
{
    $parents[$elements['id_parent']][] = $id;
}
echo "<pre>";
print_r($parents);
echo "</pre>";
?>

```

Массив `$parents` служит своеобразным справочником, запросив в котором индекс сообщения, можно узнать индексы всех непосредственных ответов на него. Дамп массива `$parents` для созданного ранее массива `$comments` представлен далее:

```

Array
(
    [0] => Array
        (
            [0] => 1
        )
    [1] => Array
        (
            [0] => 2
            [1] => 4
        )
    [2] => Array
        (
            [0] => 3
        )
    [4] => Array
        (
            [0] => 5
        )
    [3] => Array
        (
            [0] => 6
        )
)

```

Теперь можно приступить к выводу иерархии сообщений. Для этого, как правило, создают рекурсивную функцию (листинг 2.56). Для удобства и снижения издержек массивы `$comments` и `$parents` объявляются внутри функции глобальными при помощи ключевого слова `global`, что позволяет не передавать их через параметры, а использовать напрямую.

Листинг 2.56. Рекурсивная функция вывода иерархии сообщений

```

<?php
. . .
// $id_parent — индекс родительского сообщения
// $indent   — отступ
function print_hierarchy($id_parent = 0, $indent = "")
{

```

```

// Объявляем внешние массив $comments и $parents
// глобальным и доступным для использования
// внутри функции
global $comments, $parents;
// Проверяем, имеются ли у элемента с индексом
// $id_parent потомки, иначе покидаем функцию -
if(!isset($parents[$id_parent])) return;
// Выводим все элементы массива с родителем,
// имеющим индекс $id_parent
for($i = 0; $i < count($parents[$id_parent]); $i++)
{
    // Выводим сообщение с отступом
    echo "<div style='padding-left:{$indent}px'>".
    $comments[$parents[$id_parent] [$ i1] ['text']]."</div>";
    // Выводим иерархию ответов для текущего // сообщения
    print_hierarchy($parents[$id_parent] [$i], $indent + 20);
}
}
// Выводим сообщения, начиная с первого
print_hierarchy();
?>

```

Функция `print_hierarchy()` из листинга 2.56 принимает два необязательных параметра, первый из которых `$id_parent` передает индекс сообщения. Если параметр `m` утипн или принимает значение 0, вывод начинается с первого сообщения иерархии. Второй параметр `$indent` передает целое число пикселей, на которое необходимо сдвинуть ответы вправо относительно родительского сообщения. Сдвиг осуществляется при помощи CSS-параметра `padding-left`. Для каждого из выводимых сообщений, функция вызывает сама себя, передавая новый индекс и увеличивая значения отступа на 20 пикселей. Результат выполнения скрипта из листинга 2.56 представлен на рис. 2.3.

Постраничная навигация

Количество элементов в массиве может принимать огромные значения. Для того чтобы предоставить пользователю возможность удобно просматривать значения при выводе содержимого массива на печать, часто прибегают к разбиению результата на отдельные страницы. Организацию набора ссылок для удобного просмотра такого набора страниц называют *постраничной навигацией*.

Пусть имеется массив, содержащий список языков программирования (листинг 3.62), причем список необходимо вывести таким образом, чтобы на одной странице размещалось по два элемента массива. Для решения этой задачи необходимо весь массив разбить на пары и передавать номер текущей пары через GET-параметр (в примере использован GET-параметр с именем `page`). В зависимости от переданного номера скрипт вычисляет начальный

`$start` и конечный `$end` индексы, между которыми необходимо вывести элементы массива на текущей странице.

Листинг 2.57. Постраничная навигация

```
<?php
    $language[] = "PHP";
    $language[] = "C++";
    $language[] = "Java";
    $language[] = "Ruby";
    $language[] = "Python";
    $language[] = "Perl";
    $language[] = "Visual Basic";
    $language[] = "Fortran";
    $language[] = "Pascal";
    $language[] = "Assembler";
    $language[] = "Lisp";
    $language[] = "Haskell";
    $language[] = "C#";

    // Определяем количество элементов на одной странице
    $pnumber = 2;

    // Проверяем, передан ли номер текущей страницы
    if(isset($_GET['page'])) $page = intval($_GET['page']) ;
    else $page = 1;

    // Количество элементов в массиве
    $total = count($language);
    // Вычисляем количество страниц
    $number = (int)($total/$pnumber);
    if((float)($total/$pnumber) - $number != 0) $number++;

    // Начальный индекс массива $language
    // для вывода на текущей странице
    $start = (($page - 1)*$pnumber + 1);
    // Конечный индекс массива $language
    // для вывода на текущей странице
    $end = $page*$pnumber + 1;
    if($end > $total) $end = $total;

    // Выводим содержимое страниц
    for($i = $start; $i < $end; $i++)
    {
        echo $language[$i]."<br />";
    }

    // Постраничная навигация for($i = 1; $i <= $number; $i++)
    {
        // Если это произвольная страница
        if($i != $number)
        {
            if($page == $i)
            {
                // Текущую страницу не подсвечиваем ссылкой
                echo " [".( ($i - 1)*$pnumber + 1). "-".$i*$pnumber."&nbsp;";
            }
        }
    }
}
```

```

else
{
    echo "<a href='index.php?page=$i'>[".
        (( $i - 1 ) * $pnumber + 1). "-" . $i * $pnumber ."]</a>&nbsp;";
}
}
// Если это последняя страница, заменяем последнюю цифру
// максимальным числом позиций в массиве $temp
else
{
    if( $page - $i )
    {
        // Текущую страницу не подсвечиваем ссылкой
        echo "[" . (( $i - 1 ) * $pnumber + 1). "-" . ( $total - 1 ). "]" &nbsp;";
    }
    else
    {
        echo "<a href='index.php?page=$i'>[".
            (( $i - 1 ) * $pnumber + 1). "-" . ( $total - 1 ). "]" </a>&nbsp;";
    }
}
}
}
?>

```

Результат работы скрипта из листинга 2.57 представлен на рис. 2.4.

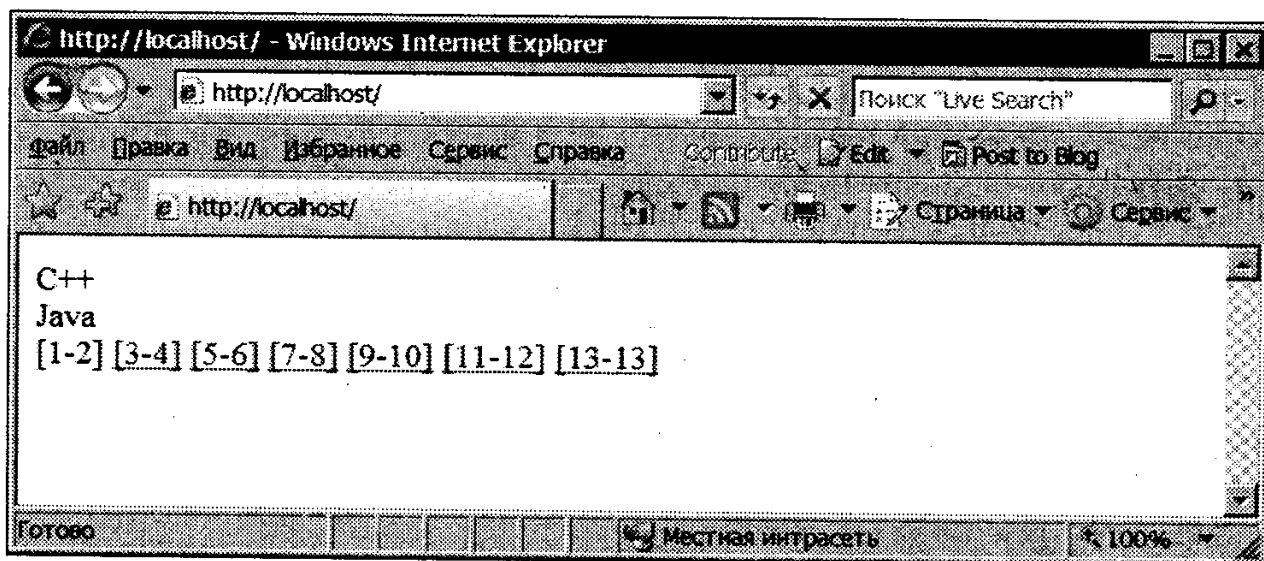


Рисунок 2.4 – Постраничная навигация

Индивидуальное задание

1 Создайте индексный массив

- Выведите информацию 2 способами
- Отсортируйте массив 3 способами
- Произведите поиск по категории

- Произведите поиск по значению
- Выведите массив постранично
- Измените массив так, чтобы данные можно было вывести в виде иерархической структуры. Осуществите вывод

2 Создайте ассоциативный/комбинированный массив

- Выведите информацию 2 способами
- Отсортируйте массив 3 способами
- Произведите поиск по категории
- Произведите поиск по значению

Вариант 1

В массиве хранятся следующие данные об учениках: фамилия, имя, отчество, рост, масса. Вычислить средний рост учеников, рост самого высокого и самого низкого ученика. Сколько учеников могут заниматься в баскетбольной секции, если рост баскетболиста должен быть больше 170 см?

Создайте ассоциативный массив, содержащий названия книг, организованных по жанрам: («детектив», "женский роман", "классика" и др.), а элементами — названия книг.

Вариант 2

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

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

Вариант 3

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

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

Вариант 4

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

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

Вариант 5

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

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

**Лабораторная работа №11
Создание функций на PHP**

Цель работы: Получение навыков создания функций

Порядок выполнения работы.

Изучить теоретические сведения.

Выполнить задание к лабораторной работе в соответствии с вариантом.

Оформить отчет.

Требования к отчету.

Цель работы.

Постановка задачи.

Текст программы.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Создания функции в PHP:

В PHP объявление функции очень простое, вам просто надо написать слово `function`, после него название функции, круглые скобки, фигурные скобки и уже в них писать код.

Вызвать её тоже очень просто, пишем название функции и круглые скобки.

```

PHP
1      // Объявление (создание) функции с названием hello_world
2      function hello_world () {
3          echo 'Hello World</br>';
4      }
5
6      // Вызов (запуск) функции hello_world
7      hello_world();

```

Также, если мы её вызовим несколько раз, то получим несколько предложений «Hello World».

```

PHP
1      // Объявление функции с названием hello_world
2      function hello_world () {
3          echo 'Hello World</br>';
4      }
5
6      // Вызов несколько раз функции hello_world
7      hello_world();
8      hello_world();
9      hello_world();

```

Параметры в функции:

Также в PHP есть передача параметров в функции, это очень полезно, к примеру когда вам нужно сравнить два числа, вы просто передаёте эти числа в функцию и она уже сравнивает.

```

PHP
1      // Объявление функции с названием num_top
2      function num_top ($a, $b) {
3          if ($a > $b) {
4              echo '$a > $b, $a = ' . $a . '</br>';
5          } elseif ($a < $b) {
6              echo '$a < $b, $b = ' . $b . '</br>';
7          } else {

```

```

8         echo '$a = $b, $b и $a = ' . $a . '</br>';
9     }
10 }
11
12 // Вызов несколько раз функции num_top с разными
13 параметрами
14     num_top(3, 1); // параметр $a = 3, параметр $b = 1
15     num_top(2, 9); // параметр $a = 2, параметр $b = 9
16     num_top(5, 5); // параметр $a = 5, параметр $b = 5

```

Также в качестве параметра можно передавать переменную, при чём, вообще без разницы какую, тут всё зависит от того, что вам нужно делать в функции, можно даже передавать массивы.

Возврат значения в функции:

Также функция может возвращать разные значения, это нужно, к примеру, если нам не надо, чтобы функция выводила на экран, какое значение больше, а только давала нам это значение, для дальнейшей работы с ним.

Для этого, внутри функции используется ключевое слово `return`.

```

PHP
1     // Объявление функции с названием num_top
2     function num_top ($a, $b) {
3         if ($a >= $b) {
4             // Возвращает $a, если $a больше или равно $b
5             return $a;
6         } elseif ($a < $b) {
7             // Возвращает $b, если $a меньше $b
8             return $b;
9         }
10    }
11
12 // Вызов несколько раз функции num_top с разными
13 параметрами
14     $a = num_top(3, 1); // Переменная $a равна 3
15     $b = num_top(5, 10); // Переменная $a равна 10
16
17 // выводим на экран, чему равны переменные
18 echo '$a = ' . $a . ', $b = ' . $b . '</br>';
19 // Складываем их и выводим на экран
20 echo $a + $b;

```

Как видите всё работает, сначала сравниваем три и один, получается три больше чем один, число возвращается и присваивается переменной \$a, точно также с переменной \$b, после этого складываем эти две переменный и получается 13.

Ещё в PHP можно передать функцию как параметр, это иногда бывает полезно, что бы не создавать отдельную переменную.

```

PHP
1      // Объявление функции с названием num_top
2      function num_top ($a, $b) {
3          if ($a >= $b) {
4              // Возвращает $a, если $a больше или равно $b
5              return $a;
6          } elseif ($a < $b) {
7              // Возвращает $b, если $a меньше $b
8              return $b;
9          }
10     }
11
12     // Вызываем функцию, и передаём другую функцию
13     echo num_top(num_top(3, 2), num_top(1, 4)); // Выведет 4

```

Как видите всё просто, это пожалуй всё основное что нужно знать о функциях, единственное, примеры были достаточно однообразны.

Глобальные и статические переменные:

Это часть не очень сильно относится к функциям, но тем не менее её надо знать.

Глобальные переменные:

Для начала рассмотрим глобальные переменные, это переменные которые объявлены вне какой функции, то есть внутри функции их нет, их называют глобальными, а те которые объявлены внутри функции, называются локальными.

В том дела, что если вы в функции хотите использовать глобально переменную, то вы не сможете это так просто сделать.

Объявление глобальной переменной в PHP, очень простое, просто объявляете её вне функции, вот получить её будет тоже очень просто, для этого надо использовать ключевое слово `global` или массив `$GLOBALS`, рассмотрим оба эти варианта.

```

PHP
1      $a = 10; // Объявление глобальной переменной
2

```

```

3      // Объявление функции с названием plus
4      function plus () {
5          $b = 5; // Объявление локальной переменной
6          global $a; // Получение глобальной переменной
7          echo $a + $b; // Складываем и выводим на экран
8
9      }
10
11     // Вызываем функцию plus
12     plus();

```

Как видите функция взяла переменную \$a, которая равна 10, и сложила её с внутренней переменной \$b, которая равна 5, вывела на экран 15.

Теперь рассмотрим массив \$GLOBALS, но это просто массив, который хранит в себе все глобальные переменные.

Важно: для названия переменной, внутри квадратных скобок, используется строка названия переменной без знака доллара в начале.

```

PHP
1      $a = 10; // Объявление глобальной переменной
2
3      // Объявление функции с названием plus
4      function plus () {
5          $b = 5; // Объявление локальной переменной
6          echo $GLOBALS['a'] + $b; // Складываем и выводим на
7 экран
8
9      }
10
11     // Вызываем функцию plus
12     plus();

```

Вывод будет точно такой же, как из первого примера,

Ну а что если мы просто напишем \$a + \$b?

То тогда он просто выведет 5, так как, PHP создаёт новую локальную переменную \$a, которая равна нулю, потому что ей нечего не присваивается.

Статические переменные:

Статичная переменная в PHP, это ещё очередной тип переменной, как глобальная или локальная.

По сути это локальная переменная, но различие в том, что если создаём её, то она не удаляется, после того как функция закончит свою работу.

Что бы создать статичную переменную, надо перед её объявлением использовать ключевое слово `static`.

```

PHP
1      // Объявление функции с названием plus_one
2      function plus_one () {
3          static $a = 0; // Объявление статичной переменной $a
4          $a++; // увеличение переменной на один
5          echo $a . " "; // Вывод переменной на экран
6      }
7
8      // вызываем функцию plus_one
9      for ($i = 0; $i < 10; $i++) {
10         plus_one();
11     }

```

Просто выводит единицы, потому что при каждом вызове функции, переменная `$a`, удаляется и создаётся заново, статичная переменная нет.

Вывод:

В этой части вы узнали о том как происходит создания функции в PHP 7 и вообще что это такое, также, узнали о глобальных и статичных переменных.

1 Глобальные массивы

В PHP версии 4 и выше введено понятие "суперглобальных" массивов. Эти массивы содержат всю информацию о состоянии сервера и среды выполнения скрипта. Массивы доступны в любом месте скрипта без дополнительных объявлений, т.е. не надо использовать ключевое слово **global**.

Всего массивов девять. Имена всех массивов записываются заглавными буквами, а начинается имя всегда с "\$_" (кроме массива \$GLOBALS).

<code>\$GLOBALS</code>	Массив содержит ссылки на все переменные, объявленные в данном скрипте. Это ассоциативный массив, в котором имена переменных являются ключами.
<code>\$_SERVER</code>	Массив содержит все данные о настройках среды выполнения скрипта и параметры сервера.
<code>\$_GET</code>	Список переменных, переданных скрипту методом GET, т.е. через параметры URL-запроса.
<code>\$_POST</code>	Список переменных, переданных скрипту методом POST.
<code>\$_COOKIE</code>	Массив содержит все cookies, которые сервер установил на стороне пользователя.
<code>\$_FILES</code>	Содержит список файлов, загруженных на сервер из формы. Более детально мы рассмотрим этот массив в уроке, посвящённом загрузке файлов на сервер.
<code>\$_ENV</code>	Содержит переменные окружения, установленные для всех скриптов на сервере.
<code>\$_REQUEST</code>	Этот массив объединяет массивы <code>\$_GET</code> , <code>\$_POST</code> и <code>\$_COOKIE</code> . очень часто бывает удобен при обработке пользовательских запросов, но применять его для защищённой обработки данных не стоит.
<code>\$_SESSION</code>	Массив содержит все переменные сессии текущего пользователя.

Просмотреть содержимое всех массивов можно в результате вызова функции `phpinfo()`.

Рассмотрим примеры использования глобальных массивов.

`$_SERVER`

С помощью этого массива можно узнать практически всё о сервере, на котором выполняется скрипт. Например:

```
<?php
echo 'параметры сервера:' . "<br />\n";

echo "Операционная система: " .
    $_SERVER["OS"] . "<br />\n";
echo "Web-сервер: " .
    $_SERVER["SERVER_SOFTWARE"] . "<br />\n";
echo "Имя сервера: " .
    $_SERVER["SERVER_NAME"] . "<br />\n";
```

```

echo "Адрес сервера: " .
    $_SERVER["SERVER_ADDR"] . "<br />\n";
echo "Порт сервера: " .
    $_SERVER["SERVER_PORT"] . "<br />\n";
echo "Адрес клиента: " .
    $_SERVER["REMOTE_ADDR"] . "<br />\n";
echo "Путь к документам на сервере: " .
    $_SERVER["DOCUMENT_ROOT"] . "<br />\n";
echo "Полный путь к текущему скрипту: " .
    $_SERVER["SCRIPT_FILENAME"] . "<br />\n";
echo "Имя текущего скрипта: " .
    $_SERVER["PHP_SELF"] . "<br />\n";

```

?>

Пользоваться этим массивом нужно аккуратно, т.к. некоторые сервера имеют достаточно специфичные настройки или значения параметров. Иногда это приводит к неприятным ошибкам, которые к тому же очень сложно диагностировать и исправить. Например, параметры `$_SERVER["REQUEST_URI"]` и `$_SERVER["SCRIPT_NAME"]` могут быть не установлены (хотя один из них как правило присутствует).

`$_GET` и `$_POST`

В эти массивы помещаются данные, передаваемые скрипту извне (так называемый, пользовательский ввод). В принципе, пользователь может влиять только на эти два массива плюс массив файлов и cookie. И именно поэтому все элементы этих массивов должны тщательно проверяться на допустимые значения.

Например, если пользователь ввёл в строку адреса браузера адрес `"http://localhost/index.php?name=Igor&fam=Sazonow&jt=ректор&wuz=BRU"`, то массив `$_GET` надо будет обрабатывать так:

```

<?php
/*
    Предполагаем, что массив $_GET должен
    содержать следующие элементы

    $_GET['name'] = "Igor";
    $_GET['fam'] = "Sazonow ";
    $_GET['jt'] = "ректор";
    $_GET['wuz'] = "BRU";
*/

// Теперь проверим наличие данных,
// а для недостающих - возьмём пустую строку

$name = (isset($_GET['name']))?
    $_GET['name']:' не указано ';
$fam = (isset($_GET['fam']))?
    $_GET['fam']:' не указано ';
$jst = (isset($_GET['jt']))?
    $_GET['jt']:' не указано ';
$country = (isset($_GET['country']))?
    $_GET['wuz']:' не указано ';

echo "Вуз: $wuz <br />\n";
echo "Фамилия: $fam <br />\n";
echo "Имя: $name <br />\n";
echo "Должность: $jst <br />\n";

```

?>

Этот скрипт будет устойчиво работать при любом GET-запросе, даже если не будет указан ни один параметр. Если используется метод POST, то достаточно заменить \$_GET на \$_POST. А если вам совершенно не важно, откуда получены данные - воспользуйтесь массивом \$_REQUEST (см. ниже).

Все вводимые данные должны проверяться на корректность.

Примечание

isset

(PHP3, PHP4, PHP5)

isset - определяет, установлена ли переменная.

Описание

bool **isset** (*mixed* var [, *mixed* var [, ...]])

Примечание: **isset()** это конструкция языка.

Возвращает **TRUE**, если var существует; иначе **FALSE**.

Если переменная была разустановлена/unset с помощью функции [unset\(\)](#), она больше не сможет быть **isset()**. **isset()** возвратит **FALSE**, если проверяет переменную, которая была установлена **NULL**. Также отметьте, что **NULL**-байт ("\0") не является эквивалентом PHP-константы **NULL**.

```
<?php
$a = "test";
$b = "anothertest";

echo isset ($a); // TRUE
echo isset ($a, $b); //TRUE

unset ($a);
echo isset ($a); // FALSE
echo isset ($a, $b); //FALSE

$foo = NULL;
print isset ($foo); // FALSE
?>
```

Это также работает с элементами массивов:

```
<?php
$a = array ('test' => 1, 'hello' => null);

echo isset ($a['test']); // TRUE
echo isset ($a['foo']); // FALSE
echo isset ($a['hello']); // FALSE
echo array_key_exists('hello', $a); // TRUE
?>
```

\$_COOKIES

В массив \$_COOKIES автоматически помещаются все cookies, которые получены от браузера. Механизм cookies подробно будет рассмотрен в соответствующей лабораторной.

\$_REQUEST

Массив `$_REQUEST` объединяет три массива: `$_POST`, `$_GET`, `$_COOKIES`. В ранних версиях PHP сюда же входил массив `$_FILES`, но из соображений безопасности и производительности его исключили. Пользоваться массивом `$_REQUEST` очень удобно в случаях, когда нет разницы, каким методом был сделан запрос. Например, при страничном выводе данных номер страницы можно передавать через URL ("`index.php?page=3`") методом GET, а можно через элемент формы методом POST. И в том и в другом случае переменная **page** попадёт в массив `$_REQUEST`.

\$_SESSION

Предназначение массива `$_SESSION` - хранение всех переменных сессии текущего пользователя.

Глобальные массивы очень удобны в работе, но злоупотреблять ими не стоит, особенно массивом `$_SESSION`. Если переменной нужна только в локальной области видимости - не надо её делать глобальной. Слишком большое количество "лишних" переменных сильно снижает быстродействие и эффективность скрипта.

2 Обработка формы в PHP

Достаточная доля задач в PHP связана с обработкой данных, полученных от пользователя. И в большинстве случаев, эти данные получены из формы, поэтому **обработка форм в PHP** является важнейшим моментом при **создании сайта**.

Создадим **форму на HTML**:

```
<html>
<head>
<title>Форма</title>
</head>
<body>
    <form name = 'myform' action = 'req.php' method = 'post'>
        Ваш логин: <input type = 'text' name = 'login' />
<br />
        Ваш пароль: <input type = 'password' name = 'pass' />
<br />
<input type = 'submit' value = 'Войти' />
</form>
    </body>
</html>
```

Это пример классической **формы авторизации пользователя**, которую, все многократно заполняли на самых разных сайтах.

Рассмотрим как отправляются данные. Рассмотрим два основных метода - **GET** и **POST**. Главное их отличие - это вид отправки: открытый (**GET**) и закрытый (**POST**). Самый лучший способ понять, в чём разница - это открыть данный **HTML-код** в браузере и нажать кнопку "**Войти**". Посмотрите на адресную строку, а потом поставьте другой метод, снова пройдите и вновь посмотрите на адресную строку. Отличие будет бросаться в глаза мгновенно.

Соответственно, в **PHP** созданы два массива: **\$_GET** и **\$_POST**, которые содержат данные, полученные каждым из этих методов. Также есть массив **\$_REQUEST**, который содержит данные **\$_GET** и **\$_POST** одновременно.

Чаще всего используется метод **POST** и массив **\$_POST**, хотя, безусловно, зависит от ситуации, но в большинстве случаев делают именно так.

Реализуем простой скрипт (в файле "**req.php**"):

```
<?php
$login = $_POST['login'];
$pass = $_POST['pass'];
if (($login == "Admin") && ($pass == "AdminPass"))
    echo "Привет, Admin!";
else echo "Доступ закрыт";
?>
```

В данном скрипте мы получаем данные, полученные из формы методом **POST** (из массива **\$_POST**, хотя с таким же успехом могли получить эти данные из **\$_REQUEST**). Далее проверяем логин и пароль и выводим: "Привет, Admin!" или "Доступ закрыт". Разумеется, чтобы создать систему авторизации пользователей на сайте, необходимо ещё узнать о **cookie** или о **сессиях** (в принципе, тоже **cookie**).

Аналогично, **считываются и обрабатываются абсолютно любые данные из форм**. То есть всё, что нужно - это знать имя переменной (задаётся в **HTML-форме**) и дальше использовать массивы **\$_POST**, **\$_GET** и **\$_REQUEST**.

3 Проверка данных формы с помощью PHP

Рассмотрим пример с еще одной классической формой. Она создана следующим образом:

```
<form action="app/check.php" method="post">
<p>Имя: <input name="name" type="text"></p>
<p>Фамилия: <input name="surname" type="text"></p>
<p>E-mail: <input name="email" type="text"></p>
<p>Сообщение: <br /><textarea name="message" cols="30"
rows="5"></textarea></p>
<p><input type='submit' value='Отправить'></p>
</form>
```

1. Создайте и откройте (через редактор) обработчик `app/code.php` - сначала, это просто пустая страница. Далее откройте тег PHP - `<?php`.

2. Сначала, нужно проверить была ли отправлена форма, для этого мы будем использовать глобальную переменную `$_SERVER` и проверять `REQUEST_METHOD`

```
<?php
if($_SERVER['REQUEST_METHOD'] == 'POST') {
    // наш код
}
?>
```

3. Далее, если форма отправлена, мы можем получить данные от поля "Имя", для этого, какой-нибудь переменной присваиваем полученное значение от этого поля, например:

```
<?php
$name = $_POST['name'];
?>
```

`$_POST` - переменная, в которой сохраняются данные, если форма была отправлена методом POST.

`$_POST['name']` - получаем данные от поля **name**.

4. Тоже самое делаем и для остальных полей:

```
<?php
$name = $_POST['name'];

$surname = $_POST['surname'];
$email = $_POST['email'];
$message = $_POST['message'];
?>
```

5. Данные мы получили, теперь мы можем их вывести, для этого в страницу обработчика дописываем код:

```
<?php
$name = $_POST['name'];

$surname = $_POST['surname'];
$email = $_POST['email'];
$message = $_POST['message'];

echo $name."<br />".$surname."<br />".$email."<br />".$message."<br />";
?>
```

будем проверять на корректность вводимых данных, так как если создать форму без такой проверки, то пользователи могут навредить сайту.

Введем проверку корректности:

1. Используем ту часть кода, где мы получили данные из формы:

```
<?php
$name = $_POST['name'];
$surname = $_POST['surname'];
$email = $_POST['email'];
$message = $_POST['message'];
?>
```

2. Теперь нам нужно проверить переданные нам данные. Чтобы не писать один и тот же код, давайте создадим несколько функций для проверки.

Сначала создадим функцию для очистки данных от HTML и PHP тегов:

```
<?php
function clean($value = "") {
    $value = trim($value);
    $value = stripslashes($value);
    $value = strip_tags($value);
    $value = htmlspecialchars($value);

    return $value;
}
?>
```

Здесь, мы использовали функцию `trim` для удаления пробелов из начала и конца строки.

Функция `stripslashes` нужна для удаления экранированных символов ("Вас зовут O'reilly?" => "Вас зовут O'reilly?").

Функция `strip_tags` нужна для удаления HTML и PHP тегов. Последняя функция - `htmlspecialchars` преобразует специальные символы в HTML-сущности ('&' преобразуется в '&' и т.д.)

Дальше, создадим функцию для проверки длины строки:

```
<?php
function check_length($value = "", $min, $max) {
    $result = (mb_strlen($value) < $min || mb_strlen($value) > $max);
    return !$result;
}
?>
```

Здесь, мы использовали функцию `mb_strlen` для проверки длины строки. Первый параметр, `$value` это строка, которую нужно проверить, второй параметр `$min` минимально допустимая длина строки, третий параметр `$max` - максимально допустимая длина. Если длина строки будет удовлетворительна, то функция вернет TRUE иначе FALSE.

3. Нужно обработать переменные с помощью этих функций:

```
<?php
$name = clean($name);
$surname = clean($surname);
$email = clean($email);
$message = clean($message);

if(!empty($name)    &&    !empty($surname)    &&    !empty($email)    &&
!empty($message)) {
```



```

    ...
}
?>

```

Если значения не пустые (проверили с помощью функции `empty`), то можно продолжать проверку дальше:

```

<?php
if(!empty($name)    &&    !empty($surname)    &&    !empty($email)    &&
!empty($message)) {
    $email_validate = filter_var($email, FILTER_VALIDATE_EMAIL);

    if(check_length($name, 2, 25) && check_length($surname, 2, 50) &&
check_length($message, 2, 1000) && $email_validate) {
        ...
    }
}
?>

```

Если длина значений переменных нас устраивает, то можем продолжать. Также, мы использовали функцию `filter_var` с параметром `FILTER_VALIDATE_EMAIL` для валидации электронного адреса.

4. Добавим сообщение об успешности операции, если данные прошли все проверки.

```

<?php
if(!empty($name)    &&    !empty($surname)    &&    !empty($email)    &&
!empty($message)) {
    $email_validate = filter_var($email, FILTER_VALIDATE_EMAIL);

    if(check_length($name, 2, 25) && check_length($surname, 2, 50) &&
check_length($message, 2, 1000) && $email_validate) {
        echo "Спасибо за сообщение";
    }
}
?>

```

4. В конце нужно добавить сообщения для уведомления о том, что данные не прошли проверку.

```

<?php
if(!empty($name)    &&    !empty($surname)    &&    !empty($email)    &&
!empty($message)) {
    $email_validate = filter_var($email, FILTER_VALIDATE_EMAIL);

    if(check_length($name, 2, 25) && check_length($surname, 2, 50) &&
check_length($message, 2, 1000) && $email_validate) {
        echo "Спасибо за сообщение";
    } else { // добавили сообщение
        echo "Введенные данные некорректные";
    }
} else { // добавили сообщение
    echo "Заполните пустые поля";
}
?>

```

Если все произошло успешно, то мы увидим сообщение *"Спасибо за сообщение"*

Индивидуальное задание

1 (всем) Преобразовать программу для работы с ассоциативным массивом из лабораторной №4, добавив возможность ввода данных с формы. При организации ввода реализовать максимально полную проверку корректности. Использовать функции.

2. Создать и протестировать функцию (по варианту согласно кратности номера по списку)

2.1. Среди n чисел найти наибольшее и наименьшее простые числа.

2.2. Для заданного числа n построить треугольник Паскаля.

2.3. Написать функцию, возвращающую текст приветствия в соответствии с приведенной ниже схемой

```
<?php
```

```
if((date("G") >=5)AND(date("G") <= 11 ))echo "Good Morning!";
```

```
if((date("G")>=12)ANDdate("G")<=18))echo "Good Afternoon!";
```

```
if((date("G") >= 19)AND(date("G")<= 4))echo "Good Evening!";
```

```
?>
```

Лабораторная работа №13

Введение в язык программирования Python

Цель работы: познакомиться со средой разработки Python. Изучить основные типы данных, команды ввода и вывода данных.

Краткая теория

Python – это объектно-ориентированный, интерпретируемый, переносимый язык сверхвысокого уровня. Программирование на Python позволяет получать быстро и качественно необходимые программные модули.

В комплекте вместе с интерпретатором Python идет IDLE (интегрированная среда разработки). По своей сути она подобна интерпретатору, запущенному в интерактивном режиме с расширенным набором возможностей (подсветка синтаксиса, просмотр объектов, отладка и т.п.).

Для запуска IDLE в Windows необходимо перейти в папку Python в меню “Пуск” и найти там ярлык с именем “IDLE (Python 3.X XX-bit)”.

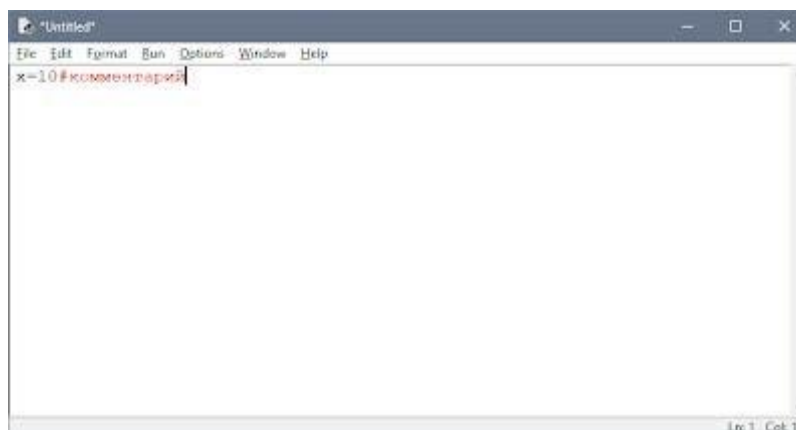
Для запуска редактора программы (кода) следует выполнить команду File->New File или сочетание клавиш Ctrl+N.

Любая Python-программа состоит из последовательности допустимых символов, записанных в определенном порядке и по определенным правилам.

Программа включает в себя:

- комментарии;
- команды;
- знаки пунктуации; • идентификаторы;
- ключевые слова.

Комментарии в Python обозначаются предваряющим их символом # и продолжаются до конца строки (т.е. в Python все комментарии являются однострочными), при этом не допускается использование перед символом # кавычек:



Знаки пунктуации

В алфавит Python входит достаточное количество знаков пунктуации, которые используются для различных целей. Например, знаки "+" или "*" могут использоваться для сложения и умножения, а знак запятой "," - для разделения параметров функций.

Идентификаторы

Идентификаторы в Python это имена используемые для обозначения переменной, функции, класса, модуля или другого объекта.

Ключевые слова

Некоторые слова имеют в Python специальное назначение и представляют собой управляющие конструкции языка.

Ключевые слова в Python:

```
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

Типы данных

1. None (неопределенное значение переменной)
2. Логические переменные (Boolean Type)
3. Числа (Numeric Type)
 1. int – целое число
 2. float – число с плавающей точкой
 3. complex – комплексное число

4. Списки (Sequence Type)
 1. list – список
 2. tuple – кортеж
 3. range – диапазон
5. Строки (Text Sequence Type)
 1. str

Ввод и вывод данных

Ввод данных осуществляется при помощи команды **input**(список ввода):

```
a = input()
```

```
print(a)
```

В скобках функции можно указать сообщение - комментарий к вводимым данным:

```
a = input ("Введите количество: ")
```

Команда `input()` по умолчанию воспринимает входные данные как строку символов. Поэтому, чтобы ввести целочисленное значение, следует указать тип данных `int()`: `a = int (input())`

Для ввода вещественных чисел применяется команда

```
a=float(input())
```

Вывод данных осуществляется при помощи команды **print**(список вывода):

```
a = 1 b = 2 print(a)
```

```
print(a + b)
```

```
print('сумма = ', a +
```

```
b)
```

Существует возможность записи команд в одну строку, разделяя их через `;`. Однако не следует часто использовать такой способ, это снижает удобочитаемость:

```
a = 1; b = 2; print(a)
```

```
print (a + b) print
```

('сумма = ', a + b) Для

команды **print** может

задаваться так

называемый

сепаратор —

разделитель между

элементами вывода:

```
x=2
```

```
y=5
```

```
print ( x, "+", y, "=", x+y, sep = " ")
```

Результат отобразится с пробелами между элементами: $2 + 5 = 7$

Простые арифметические операции над числами

$x + y$	Сложение
$x - y$	Вычитание
$x * y$	Умножение
x / y	Деление

```

Python 3.4.1: example_prost_math.py - F://Лабораторные Python/example_prost_math.py
File Edit Format Run Options Windows Help
#простейшие математические операции
x=5
y=6
print('x = ',x)
print('y = ',y)
z=x+y
print('z = ',z)
z=x-y
print('z = ',z)
z=x*y
print('z = ',z)
z=x/y
print('z = ',z)

```

Пример программы на Python

```

Python 3.4.1 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:38:22) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
x = 5
y = 6
z = 11
z = -1
z = 30
z = 0.8333333333333334
>>> |

```

Результат выполнения программы с применением простых арифметических операций

Для форматированного вывода используется **format**:

Строковый метод `format()` возвращает отформатированную версию строки, заменяя идентификаторы в фигурных скобках `{}`. Идентификаторы могут быть позиционными, числовыми индексами, ключами словарей, именами переменных.

Синтаксис команды **format**:

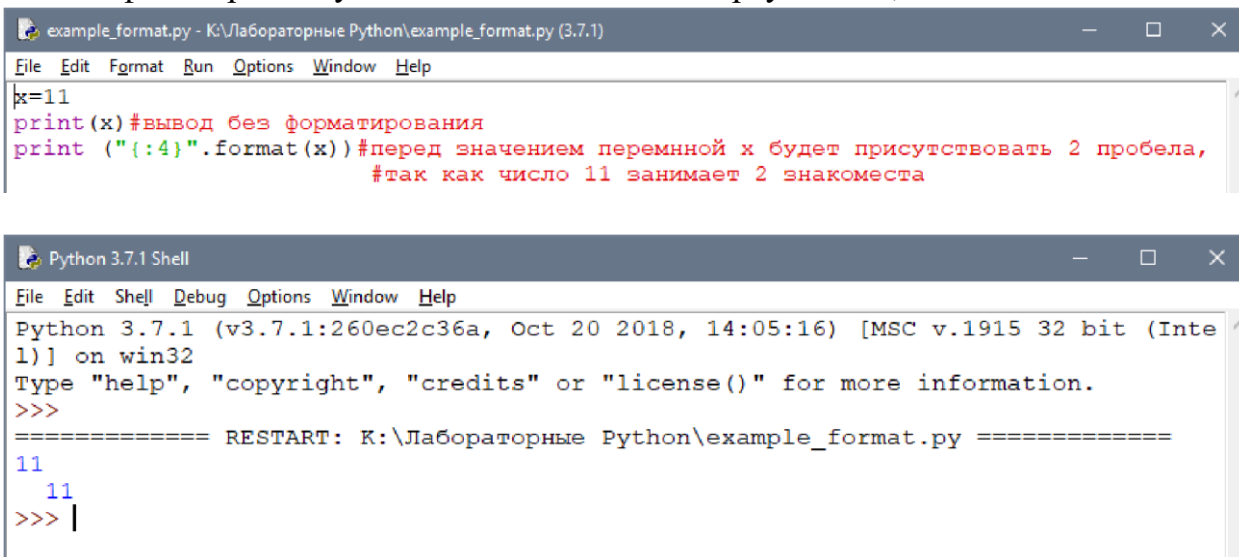
поле замены := `"{" [имя поля] ["!" преобразование] [":" спецификация] "}"`

имя поля := `arg_name ("." имя атрибута | "[" индекс "]")*`

преобразование := "r" (внутреннее представление) | "s" (человеческое представление) спецификация := см. ниже

Аргументов в format() может быть больше, чем идентификаторов в строке. В таком случае оставшиеся игнорируются.

Идентификаторы могут быть либо индексами аргументов, либо ключами:



The image shows two windows from a Python IDE. The top window, titled 'example_format.py - К:\Лабораторные Python\example_format.py (3.7.1)', contains the following code:

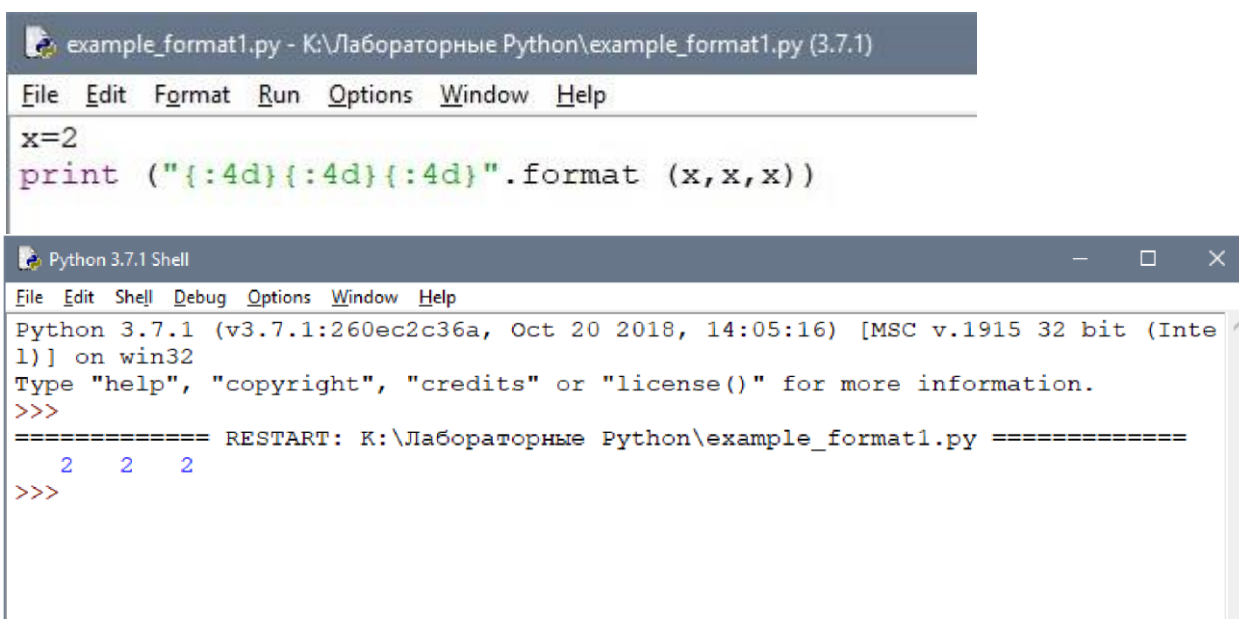
```
x=11
print(x) #вывод без форматирования
print ("{:4}".format(x)) #перед значением переменной x будет присутствовать 2 пробела,
                        #так как число 11 занимает 2 знакоместа
```

The bottom window, titled 'Python 3.7.1 Shell', shows the execution output:

```
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: К:\Лабораторные Python\example_format.py =====
11
  11
>>> |
```

В результате выведется число 11, а перед ним два пробела, так как указано использовать для вывода четыре знакоместа.

Или с несколькими аргументами:



The image shows two windows from a Python IDE. The top window, titled 'example_format1.py - К:\Лабораторные Python\example_format1.py (3.7.1)', contains the following code:

```
x=2
print ("{:4d}{:4d}{:4d}".format (x,x,x))
```

The bottom window, titled 'Python 3.7.1 Shell', shows the execution output:

```
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: К:\Лабораторные Python\example_format1.py =====
  2  2  2
>>>
```


В итоге каждое из значений выводится из расчета 4 знакоместа.

Спецификация формата:

спецификация	:= [[fill]align][sign][#][0][width][,][.precision][type]
заполнитель	:= символ кроме '{' или '}'
выравнивание	:= "<" ">" "=" "^"
знак	:= "+" "-" " "
ширина	:= integer
точность	:= integer
тип	:= "b" "c" "d" "e" "E" "f" "F" "g" "G" "n" "o" "s" "x" "X" "%" "

Тип	Значение
'd', 'i', 'u'	Десятичное число.
'o'	Число в восьмеричной системе счисления.
'x'	Число в шестнадцатеричной системе счисления (буквы в нижнем регистре).
'X'	Число в шестнадцатеричной системе счисления (буквы в верхнем регистре).
'e'	Число с плавающей точкой с экспонентой (экспонента в нижнем регистре).
'E'	Число с плавающей точкой с экспонентой (экспонента в верхнем регистре).
'f', 'F'	Число с плавающей точкой (обычный формат).
'g'	Число с плавающей точкой. с экспонентой (экспонента в нижнем регистре), если она меньше, чем -4 или точности, иначе обычный формат.
'G'	Число с плавающей точкой. с экспонентой (экспонента в верхнем регистре), если она меньше, чем -4 или точности, иначе обычный формат.
'c'	Символ (строка из одного символа или число - код символа).

's'	Строка.
'%'	Число умножается на 100, отображается число с плавающей точкой, а за ним знак %.

Для форматирования вещественных чисел с плавающей точкой используется следующая команда:

```
print('{0:.2f}'.format(вещественное число))
```

```
format_chisla.py - K:/Лабораторные Python/format_chisla.py (3.7.1)
File Edit Format Run Options Window Help
x=10
y=7
print("{0:.2f}".format(x/y))
```

В результате выведется число с двумя знаками после запятой.

```
Python 3.7.1 Shell
File Edit Shell Debug Options Window Help
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 1) ] on win32
Type "help", "copyright", "credits"
>>>
===== RESTART: K:/Лаборатор
1.43
>>> |
```

Пример

Напишите программу, которая запрашивала бы у пользователя:

- ФИО ("Ваши фамилия, имя, отчество?")
- возраст ("Сколько Вам лет?") - место жительства ("Где вы живете?")

После этого выводила бы три строки:

"Ваше имя"

"Ваш возраст"

"Вы живете в"

Решение

```
a=input('Введите ваши фамилию, имя, отчество ')
b=input('Сколько вам лет? ')
c=input('Где вы живёте? ')
print('Ваше имя ',a)
print('Ваш возраст ',b)
print('Вы живете в ',c)
```

```
Введите ваши фамилию, имя, отчество Иванов Иван Иванович
Сколько вам лет? 15
Где вы живёте? Уссурийск
Ваше имя  Иванов Иван Иванович
Ваш возраст  15
Вы живете в  Уссурийск
```

Задания для самостоятельной работы

1) Установите Python <https://www.python.org/ftp/python/3.8.5/python-3.8.5.exe>

2) Напишите программу, которая запрашивала бы у пользователя:

Имя, Фамилия, Возраст, Место жительства

- фамилия, имя ("Ваши фамилия, имя?")

- возраст ("Сколько Вам лет?") - место жительства ("Где вы живете?")

После этого выводила бы три строки:

"Ваши фамилия, имя"

"Ваш возраст"

"Вы живете в"

Лабораторная работа №14

Математические операции в Python

Цель работы: познакомиться с основными математическими операциями в Python

Язык Python, благодаря наличию огромного количества библиотек для решения разного рода вычислительных задач, сегодня является конкурентом таким пакетам как Matlab и Octave. Запущенный в интерактивном режиме, он, фактически, превращается в мощный калькулятор. В этом уроке речь пойдет об арифметических операциях, доступных в данном языке. Арифметические операции изучим применительно к числам.

Если в качестве операндов некоторого арифметического выражения используются только целые числа, то результат тоже будет целое число. Исключением является операция деления, результатом которой является вещественное число. При совместном использовании целочисленных и вещественных переменных, результат будет вещественным.

В этом уроке речь пойдет об арифметических операциях, доступных в данном языке.

Если в качестве операндов некоторого арифметического выражения используются только целые числа, то результат тоже будет целое число. Исключением является операция деления, результатом которой является вещественное число. При совместном использовании целочисленных и вещественных переменных, результат будет вещественным.

Целые числа (int)

Числа в Python 3 поддерживают набор самых обычных математических операций:

$x + y$	Сложение
$x - y$	Вычитание
$x * y$	Умножение

x / y	Деление
$x // y$	Получение целой части от деления
$x \% y$	Остаток от деления
$-x$	Смена знака числа
$abs(x)$	Модуль числа
$divmod(x, y)$	Пара $(x // y, x \% y)$
$x ** y$	Возведение в степень
$pow(x, y[, z])$	<p>x : Число, которое требуется возвести в степень. y : Число, являющееся степенью, в которую нужно возвести первый аргумент. Если число отрицательное или одно из чисел "x" или "y" не целые, то аргумент "z" не принимается. z : Число, на которое требуется произвести деление по модулю. Если число указано, ожидается, что "x" и "y" положительны и имеют тип <code>int</code>.</p>

Пример применения **ВЫШЕОПИСАННЫХ** операций над целыми числами

```

x = 5
y = 2
z = 3
x+y = 7
x-y = 3
x*y = 10
x/y = 2.5
x//y = 2
x%y = 1
-x = -5
abs(-x) = 5
divmod(x, y) = (2, 1)
x**y = 25
pow(x, y, z) = 1

```

Вещественные числа (float)

Вещественные числа поддерживают те же операции, что и целые. Однако (из-за представления чисел в компьютере) вещественные числа неточны, и это может привести к ошибкам.

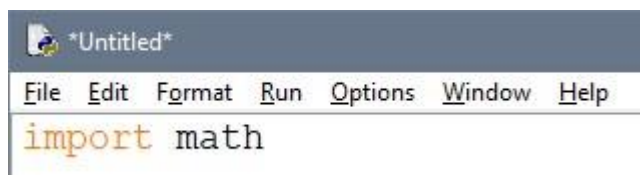
Пример применения вышеописанных операций над вещественными числами

```
x = 5.5
y = 2.3
x+y = 7.8
x-y = 3.2
x*y = 12.649999999999999
x/y = 2.3913043478260874
x//y = 2.0
x%y = 0.90000000000000004
-x = -5.5
abs(-x) = 5.5
divmod(x,y) = (2.0, 0.90000000000000004)
x**y = 50.44686540422945
```

Библиотека (модуль) math

В стандартную поставку Python входит библиотека math, в которой содержится большое количество часто используемых математических функций.

Для работы с данным модулем его предварительно нужно импортировать.



```
*Untitled*
File Edit Format Run Options Window Help
import math
```

Рассмотрим наиболее часто используемые функции модуля math

math.ceil(x)	Возвращает ближайшее целое число большее, чем x
math.fabs(x)	Возвращает абсолютное значение числа x
math.factorial(x)	Вычисляет факториал x
math.floor(x)	Возвращает ближайшее целое число меньшее, чем x
math.exp(x)	Вычисляет e^{**x}
math.log2(x)	Логарифм по основанию 2
math.log10(x)	Логарифм по основанию 10

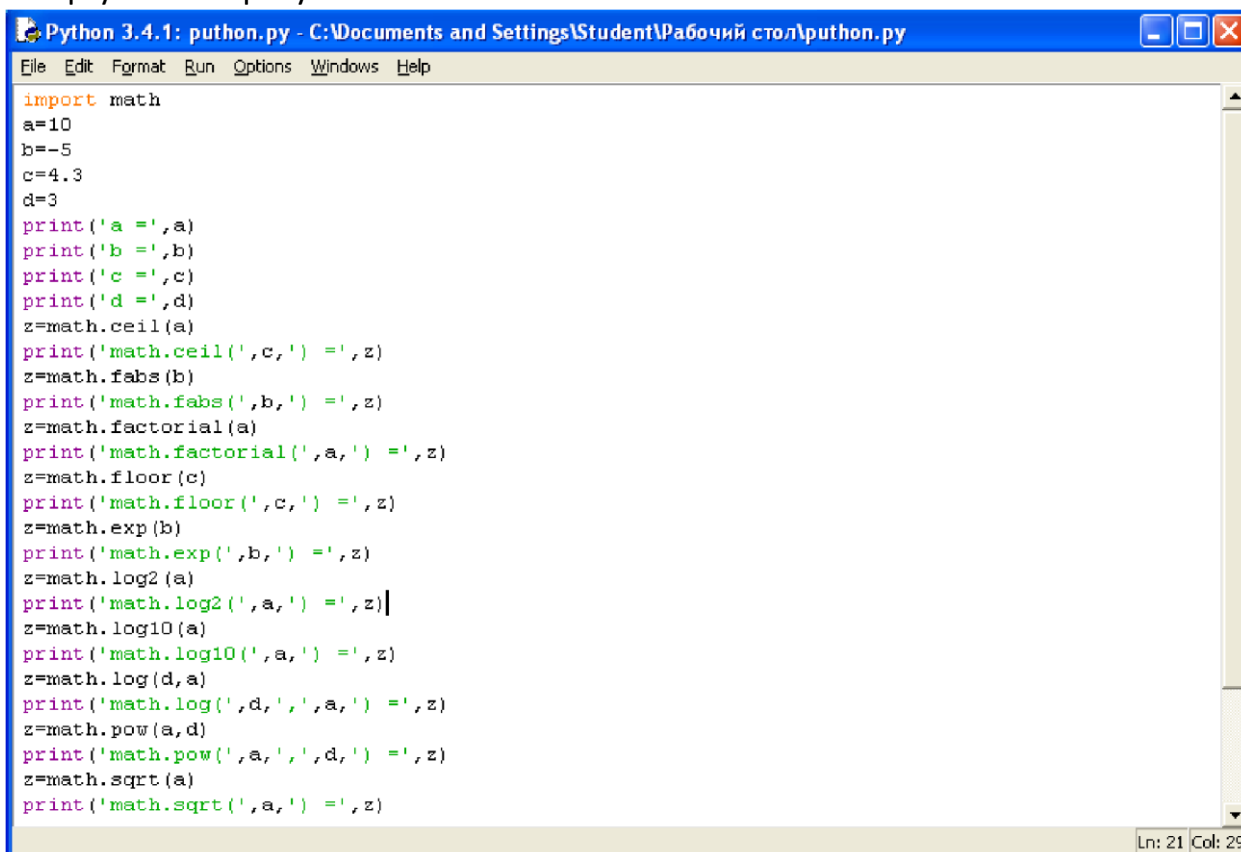
math.log(x[, base])	По умолчанию вычисляет логарифм по основанию e, дополнительно можно указать основание логарифма
math.pow(x, y)	Вычисляет значение x в степени y
math.sqrt(x)	Корень квадратный от x

Пример применения вышеописанных функций над числами В программе определены 4 переменные - a, b, c, d, каждая из которых является либо целым числом, либо вещественным, либо отрицательным.

Командой print() выводится значение каждой переменной на экран при выполнении программы.

В переменную z помещается результат выполнения функции модуля math.

Затем командой print() выводится сообщение в виде используемой функции и её аргумента и результат её выполнения.

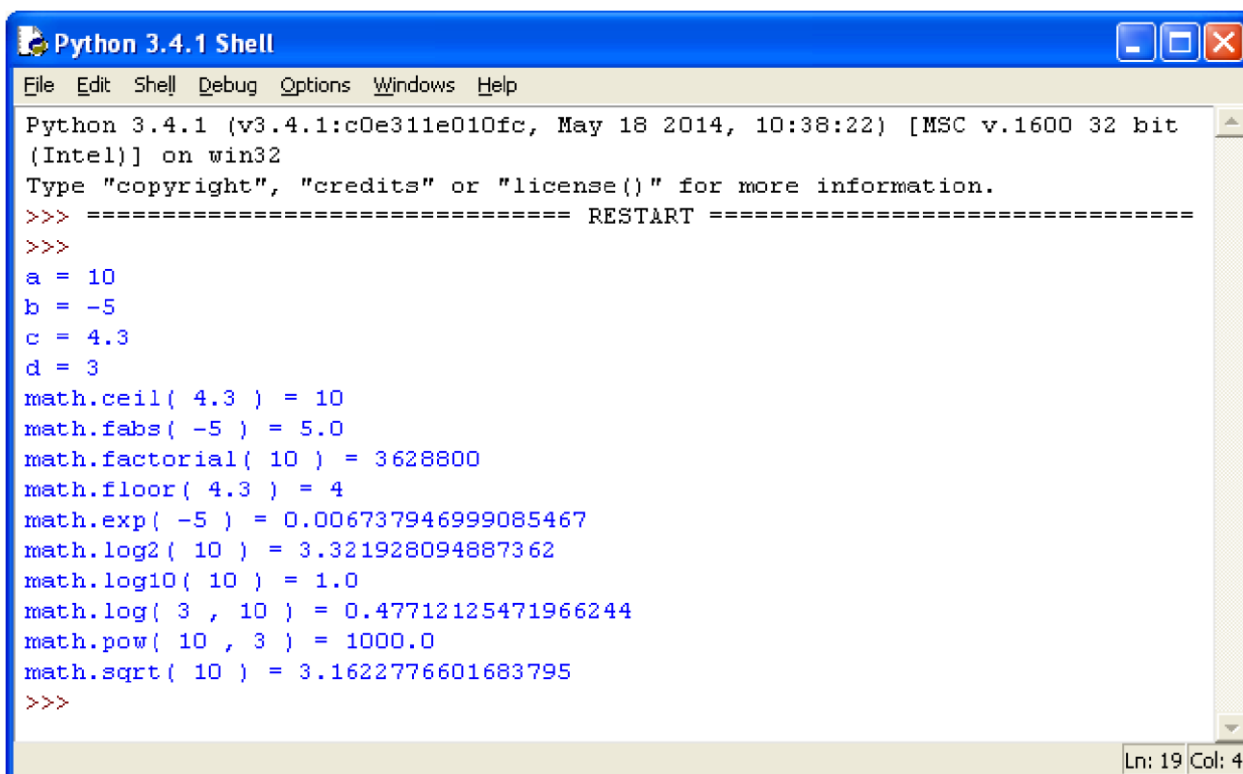


```

Python 3.4.1: puthon.py - C:\Documents and Settings\Student\Рабочий стол\puthon.py
File Edit Format Run Options Windows Help
import math
a=10
b=-5
c=4.3
d=3
print ('a =', a)
print ('b =', b)
print ('c =', c)
print ('d =', d)
z=math.ceil(a)
print ('math.ceil(', c, ') =', z)
z=math.fabs(b)
print ('math.fabs(', b, ') =', z)
z=math.factorial(a)
print ('math.factorial(', a, ') =', z)
z=math.floor(c)
print ('math.floor(', c, ') =', z)
z=math.exp(b)
print ('math.exp(', b, ') =', z)
z=math.log2(a)
print ('math.log2(', a, ') =', z)
z=math.log10(a)
print ('math.log10(', a, ') =', z)
z=math.log(d, a)
print ('math.log(', d, ',', a, ') =', z)
z=math.pow(a, d)
print ('math.pow(', a, ',', d, ') =', z)
z=math.sqrt(a)
print ('math.sqrt(', a, ') =', z)
Ln: 21 Col: 29

```

Пример программы на Python



```

Python 3.4.1 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:38:22) [MSC v.1600 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
a = 10
b = -5
c = 4.3
d = 3
math.ceil( 4.3 ) = 5
math.fabs( -5 ) = 5.0
math.factorial( 10 ) = 3628800
math.floor( 4.3 ) = 4
math.exp( -5 ) = 0.006737946999085467
math.log2( 10 ) = 3.321928094887362
math.log10( 10 ) = 1.0
math.log( 3 , 10 ) = 0.47712125471966244
math.pow( 10 , 3 ) = 1000.0
math.sqrt( 10 ) = 3.1622776601683795
>>>
Ln: 19 Col: 4

```

Результат выполнения программы с применением функций модуля math

Тригонометрические функции модуля math

math.cos(x)	Возвращает cos числа X
math.sin(x)	Возвращает sin числа X
math.tan(x)	Возвращает tan числа X
math.acos(x)	Возвращает acos числа X
math.asin(x)	Возвращает asin числа X
math.atan(x)	Возвращает atan числа X

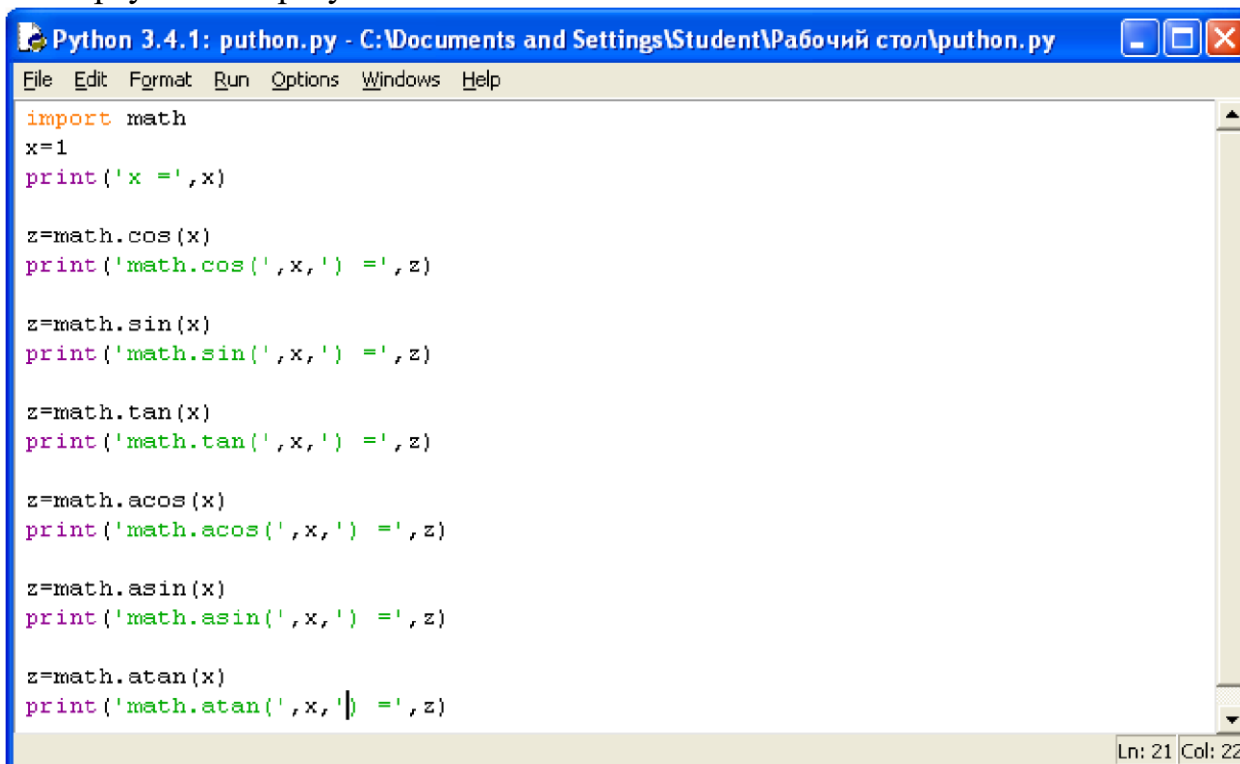
Пример применения вышеописанных функций над числами

В программе определена переменная x, содержащая целое число.

Значение переменной выводится командой print() на экран.

В переменную z помещается результат выполнения тригонометрической функции модуля `math`.

Затем командой `print()` выводится сообщение в виде используемой функции и её аргумента и результат её выполнения.



```
Python 3.4.1: puthon.py - C:\Documents and Settings\Student\Рабочий стол\puthon.py
File Edit Format Run Options Windows Help

import math
x=1
print('x =', x)

z=math.cos(x)
print('math.cos(', x, ') =', z)

z=math.sin(x)
print('math.sin(', x, ') =', z)

z=math.tan(x)
print('math.tan(', x, ') =', z)

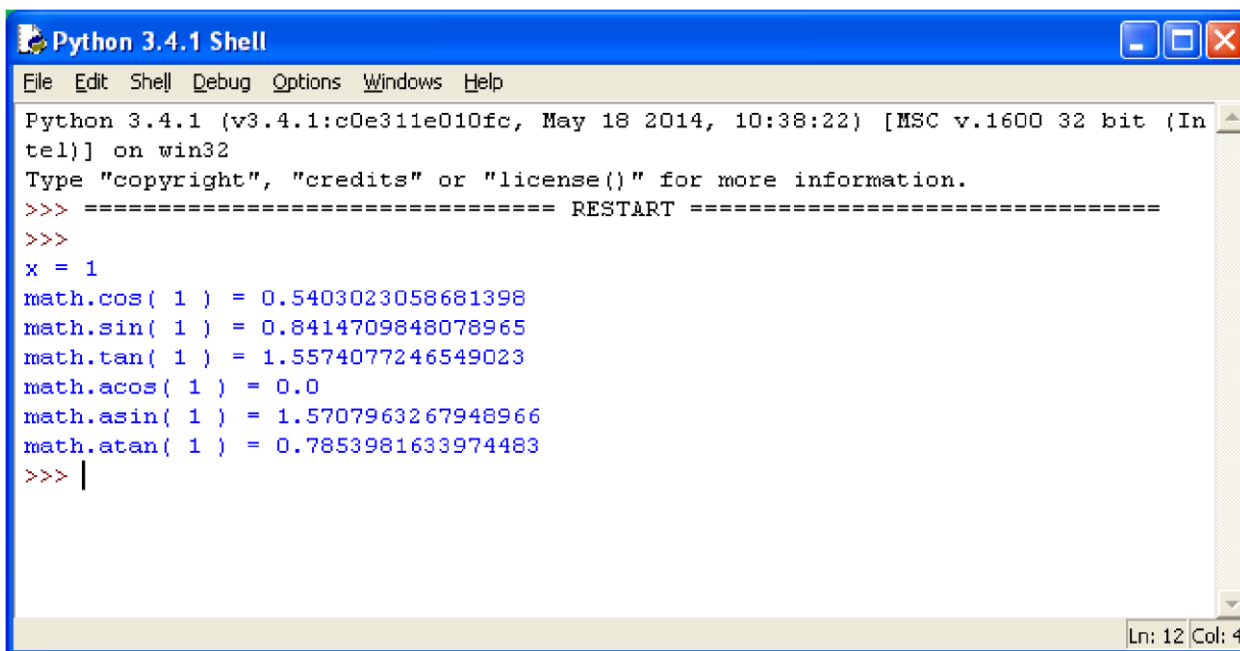
z=math.acos(x)
print('math.acos(', x, ') =', z)

z=math.asin(x)
print('math.asin(', x, ') =', z)

z=math.atan(x)
print('math.atan(', x, ') =', z)

Ln: 21 Col: 22
```

Пример программы с использованием тригонометрических функций модуля `math`



```
Python 3.4.1 Shell
File Edit Shell Debug Options Windows Help

Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:38:22) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
x = 1
math.cos( 1 ) = 0.5403023058681398
math.sin( 1 ) = 0.8414709848078965
math.tan( 1 ) = 1.5574077246549023
math.acos( 1 ) = 0.0
math.asin( 1 ) = 1.5707963267948966
math.atan( 1 ) = 0.7853981633974483
>>> |

Ln: 12 Col: 4
```

Результат выполнения программы с

применением тригонометрических функций модуля `math`

Константы:

- `math.pi` - число π .
- `math.e` - число e (экспонента).

Пример

Напишите программу, которая бы вычисляла заданное арифметическое выражение при заданных переменных. Ввод переменных осуществляется с клавиатуры. Вывести результат с 2-мя знаками после запятой.

Задание

$$Z = \frac{9\pi t + 10 \cos(x)}{\sqrt{t} - |\sin(t)|} * e^x$$

`x=10; t=1`

Решение

Сначала импортируем модуль `math`. Для этого воспользуемся командой `import math`.

Затем следует ввести значения двух переменных целого типа `x` и `t`.

Для ввода данных используется команда `input`, но так как в условии даны целые числа, то нужно сначала определить тип переменных: `x=int()`, `t=int()`.

Определив тип переменных, следует их ввести, для этого в скобках команды `int()` нужно написать команду `input()`.

Для переменной `x` это выглядит так: `x=int(input("сообщение при вводе значения"))`.

Для переменной `t` аналогично: `t=int(input("сообщение при вводе значения"))`.

Следующий шаг - это составление арифметического выражения, результат которого поместим в переменную `z`.

Сначала составим числитель. Выглядеть он будет так:

`9*math.pi*t+10*math.cos(x)`.

Затем нужно составить знаменатель, при этом обратим внимание на то, что числитель делится на знаменатель, поэтому и числитель и знаменатель нужно поместить в скобки `()`, а между ними написать знак деления `/`.

Выглядеть это будет так:

$$(9 * \text{math.pi} * t + 10 * \text{math.cos}(x)) / (\text{math.sqrt}(t) \text{math.fabs}(\text{math.sin}(t)))$$

Последним шагом является умножение дроби на экспоненту в степени x . Так как умножается вся дробь, то составленное выражение поместить в скобки $()$, а уже потом написать функцию `math.pow(math.e,x)`.

В результате выражение будет иметь вид:

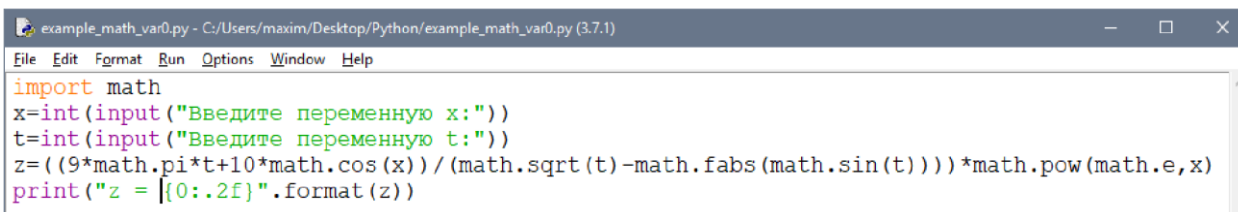
$$z = ((9 * \text{math.pi} * t + 10 * \text{math.cos}(x)) / (\text{math.sqrt}(t) \text{math.fabs}(\text{math.sin}(t)))) * \text{math.pow}(\text{math.e}, x)$$

При составлении данного выражения следует обратить внимание на количество открывающихся и закрывающихся скобок.

Командой `print()` выведем значение переменной, отформатировав его командой `format`.

Сам формат записывается в апострофах в фигурных скобках `{}`.

В задаче требуется вывести число с двумя знаками после запятой, значит вид формата будет выглядеть следующим образом: `{0:.2f}`, где `2` - это количество знаков после запятой, а `f` указывает на то, что форматируется вещественное число. При этом перед `2` нужно поставить точку, указав тем самым на то, что форматируем именно дробную часть числа.



```
example_math_var0.py - C:/Users/maxim/Desktop/Python/example_math_var0.py (3.7.1)
File Edit Format Run Options Window Help
import math
x=int(input("Введите переменную x:"))
t=int(input("Введите переменную t:"))
z=((9*math.pi*t+10*math.cos(x))/(math.sqrt(t)-math.fabs(math.sin(t))))*math.pow(math.e,x)
print("z = {}".format(z))
```

Результат



```
Python 3.7.1 Shell
File Edit Shell Debug Options Window Help
Python 3.7.1 (v3.7.1:260ec2c36a,
1) on win32
Type "help", "copyright", "credit
>>>
===== RESTART: C:/Users/maxim/
Введите переменную x:10
Введите переменную t:1
z = 2762685.71
>>> |
```

Задания для самостоятельной работы
Воспроизвести задание из примера. Сделать скриншоты кода и результата.

Лабораторная работа 3 Структура ветвление в Python

Цель работы: познакомиться со структурой ветвление (if, if-else, if-elif-else).
Научиться работать с числами и строками используя данную структуру.

Условный оператор ветвления if, if-else, if-elif-else

Оператор ветвления if позволяет выполнить определенный набор инструкций в зависимости от некоторого условия. Возможны следующие варианты использования.

1. Конструкция if

Синтаксис оператора if выглядит

так: **if логическое выражение:**

```
    команда_1  
команда_2  
    ...  
    команда_n
```

После оператора if записывается логическое выражение.

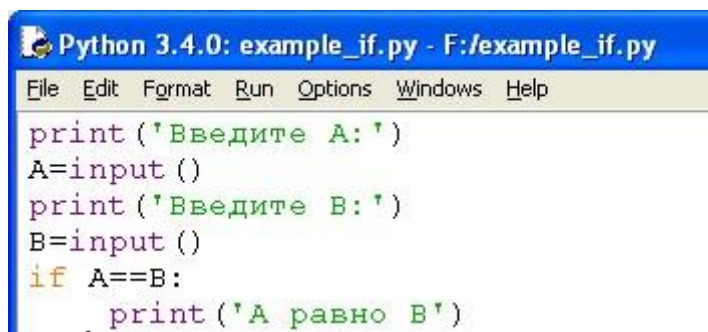
Логическое выражение — конструкция [языка программирования](#), результатом вычисления которой является «истина» или «ложь».

Если это выражение истинно, то выполняются инструкции, определяемые данным оператором. Выражение является истинным, если его результатом является число не равное нулю, непустой объект, либо логическое True.

После выражения нужно поставить двоеточие “:”.

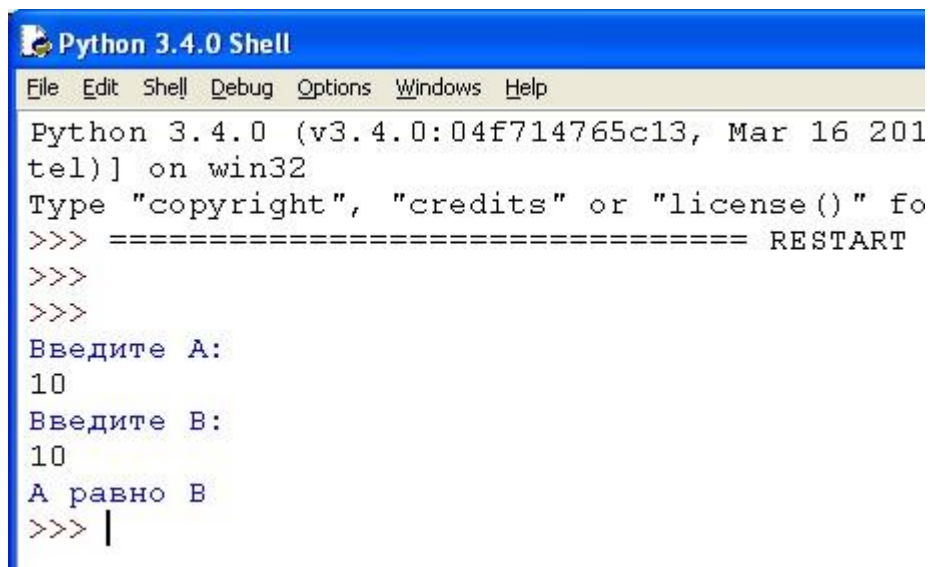
ВАЖНО: блок кода, который необходимо выполнить, в случае истинности выражения, отделяется **четырьмя** пробелами слева!

Программа запрашивает у пользователя два числа, затем сравнивает их и если числа равны, то есть логическое выражение $A==B$ истинно, то выводится соответствующее сообщение.



```
Python 3.4.0: example_if.py - F:/example_if.py
File Edit Format Run Options Windows Help
print ('Введите A:')
A=input ()
print ('Введите B:')
B=input ()
if A==B:
    print ('A равно B')
```

Пример программы на Python



```
Python 3.4.0 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.0 (v3.4.0:04f714765c13, Mar 16 201
tel)] on win32
Type "copyright", "credits" or "license()" fo
>>> ===== RESTART
>>>
>>>
Введите A:
10
Введите B:
10
A равно B
>>> |
```

Результат выполнения программы с использованием условного оператора if

2. Конструкция if – else

Бывают случаи, когда необходимо предусмотреть альтернативный вариант выполнения программы. Т.е. при истинном условии нужно выполнить один набор инструкций, при ложном – другой. Для этого используется конструкция if – else.

Синтаксис оператора if – else выглядит

так: **if** **логическое выражение:**

команда_1 команда_2

...

команда_n **else:**

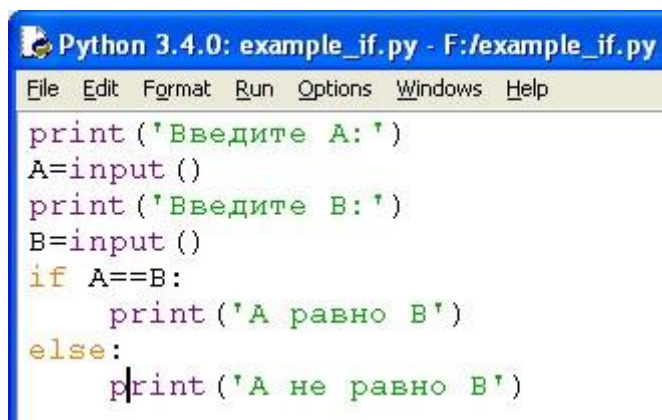
команда_1

команда_2

...

команда_n

Программа запрашивает у пользователя два числа, затем сравнивает их и если числа равны, то есть логическое выражение $A==B$ истинно, то выводится соответствующее сообщение. В противном случае выводится сообщение, что числа не равны.



```
Python 3.4.0: example_if.py - F:/example_if.py
File Edit Format Run Options Windows Help
print ('Введите A: ')
A=input ()
print ('Введите B: ')
B=input ()
if A==B:
    print ('A равно B')
else:
    print ('A не равно B')
```

Пример программы на Python

```
>>> ===== RESTART
>>>
Введите A:
10
Введите B:
5
A не равно B
>>> |
```

Результат выполнения программы с использованием условного оператора if-else

3. Конструкция if – elif – else

Для реализации выбора из нескольких альтернатив можно использовать конструкцию if – elif – else. Синтаксис оператора if – elif – else выглядит так: **if логическое выражение_1:**

команда_1

команда_2

...

команда_n **elif логическое выражение_2:**

команда_1

команда_2

...

команда_n **elif** логическое
выражение_3:

команда_1
команда_2

...

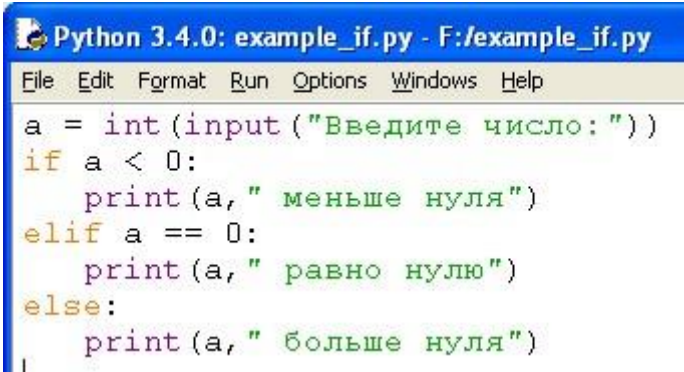
команда_n **else**:

команда_1
команда_2

...

команда_n

Программа запрашивает число у пользователя и сравнивает его с нулём $a < 0$. Если оно меньше нуля, то выводится сообщение об этом. Если первое логическое выражение не истинно, то программа переходит ко второму - $a == 0$. Если оно истинно, то программа выведет сообщение, что число равно нулю, в противном случае, если оба вышеуказанных логических выражения оказались ложными, то программа выведет сообщение, что введённое число больше нуля.



```
Python 3.4.0: example_if.py - F:/example_if.py
File Edit Format Run Options Windows Help
a = int(input("Введите число: "))
if a < 0:
    print(a, " меньше нуля")
elif a == 0:
    print(a, " равно нулю")
else:
    print(a, " больше нуля")
|
```

Пример программы на Python

```
Введите число: 41
41 больше нуля
>>> ===== RESTART =
>>>
Введите число: -5
-5 меньше нуля
>>> ===== RESTART =
>>>
Введите число: 0
0 равно нулю
>>>
```

Результат выполнения программы с использованием условного оператора if-elif-else

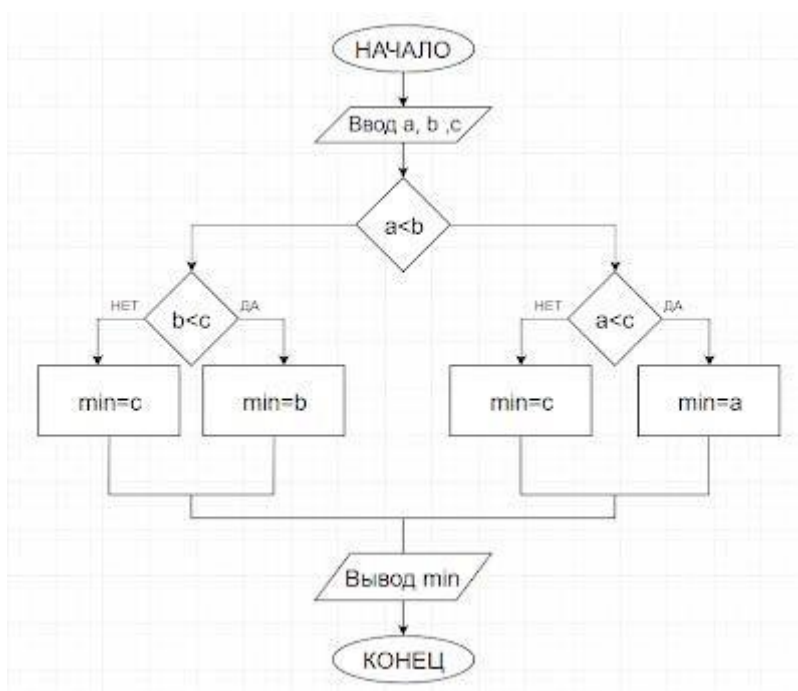
Пример

Задание

Дано 3 числа. Найти минимальное среди них и вывести на экран.

Решение

Для простоты построим блок-схему задачи.



Командами

`a=input("") b=input("") c=input("")` введём три числа, присвоив значения переменным `a`, `b`, `c`.

Условной конструкцией if-else проверим на истинность логическое выражение `a < b`. Если оно истинно, то переходим на проверку логического выражения `a < c`. Если оно истинно, то переменной "y" присвоим значение переменной "a", т.е. "a" будет минимальным, а иначе "y" присвоится значение переменной "c".

Если в начале логическое выражение `a < b` оказалось ложным, то переходим на проверку другого логического выражения `b < c`.

Если оно истинно, то "y" присвоится значение переменной "b", иначе "c".

Командой print() выводим минимальное значение.

```
#нахождение минимального из 3-х чисел
a=input('Введите целое число \n')
b=input('Введите целое число \n')
c=input('Введите целое число \n')
if a<b:
    if a<c:
        y=a
    else:
        y=c
else:
    if b<c:
        y=b
    else:
        y=c
print('Минимальное:', y)
```

Пример программы

```
Введите целое число
2
Введите целое число
5
Введите целое число
1
Минимальное: 1
```

Результат выполнения программы

Задания для самостоятельной работы

Задание

Даны три целых числа. Выбрать из них те, которые принадлежат интервалу [1,3].

Лабораторная работа №15

Ветвление и работа с цик-лами в Python

Цель работы: познакомиться с циклическими конструкциями

В Python существуют два типа циклических выражений:

- Цикл while
- Цикл for

1. Цикл while в Python

Инструкция while в Python повторяет указанный блок кода до тех пор, пока указанное в цикле логическое выражение будет оставаться истинным.

Синтаксис цикла while:

while логическое выражение:

```
команда 1  
команда 2  
...  
команда n
```

После ключевого слова while указывается условное выражение, и пока это выражение возвращает значение True, будет выполняться блок инструкций, который идет далее.

Все инструкции, которые относятся к циклу while, располагаются на последующих строках и должны иметь отступ от начала строки (4 пробела).

```
#!/ Программу по вычислению факториала  
number = int(input("Введите число: "))  
i = 1  
factorial = 1  
while i <= number:  
    factorial *= i  
    i += 1  
print("Факториал числа", number, "равен", factorial)
```

Пример программы на Python

```

Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:38:22) [MSC v.1600 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Введите число: 5
факториал числа 5 равен 120
>>> |

```

Результат выполнения программы с использованием циклического оператора while

2. Цикл for в Python:

Цикл for в Python обладает способностью перебирать элементы любого комплексного типа данных (например, строки или списка). Синтаксис цикла for: **for int in range():**

команда 1
команда 2
...
команда n

Переменной int присваивается значение первого элемента функции range(), после чего выполняются команды. Затем переменной int присваивается следующее по порядку значение и так далее до тех пор, пока не будут перебраны все элементы функции range().

Функция range() является универсальной функцией Python для создания списков (list) содержащих арифметическую прогрессию. Чаще всего она используется в циклах for.

range(старт, стоп, шаг) - так выглядит стандартный вызов функции range() в Python. По умолчанию старт равняется нулю, шаг единице.

Пример.

1. Найти сумму n элементов следующего ряда чисел: 1 -0.5 0.25 -0.125 ...
n. Количество элементов (n) вводится с клавиатуры. Вывести на экран каждый член ряда и его сумму. Решить задачу используя циклическую конструкцию for.

Решение:

В данном случае ряд чисел состоит из элементов, где каждый следующий меньше предыдущего в два раза по модулю и имеет обратный знак. Значит, чтобы получить следующий элемент, надо предыдущий разделить на -2. Какой-либо переменной надо присвоить значение первого элемента ряда (в данном случае это 1). Далее в цикле добавлять ее значение к переменной, в которой накапливается сумма, после чего присваивать ей значение

следующего элемента ряда, разделив текущее значение на -2. Цикл должен выполняться n раз.

```
n=int(input('Введите количество элементов последовательности: '))
x=1
s=0
print(x)
for i in range(n):
    s+=x
    x/=-2
    print(x)
print('Сумма ряда:',s)
```

Пример программы с циклом for

```
Введите количество элементов последовательности: 5
1
-0.5
0.25
-0.125
0.0625
-0.03125
Сумма ряда: 0.6875
```

Результат выполнения программы

2. Дано целое число, не меньшее 2. Выведите его наименьший натуральный делитель, отличный от 1.

Решение:

Для начала введём целое число командой `int(input(текст сообщения))`. Затем зададим переменной `i` значение 2. Переменная `i` выполняет роль счётчика. Если задать ей значение 1, то условие задачи не будет выполнено, а результатом всегда будет 1.

В цикле `while` в качестве логического выражения используется команда `n%i` сравниваемая с нулём. Таким образом, если остаток от деления введённого числа на текущее значение `i` не равно нулю, то счётчик увеличивается на 1, а если равно нулю цикл заканчивается и командой `print()` выводится сообщение и значение `i`.

```
n = int(input('Введите целое число не меньше 2\n'))
i = 2
while n%i != 0:
    i+=1
print('наименьший натуральный делитель:',i)
```

Пример программы с циклом while

```
Введите целое число не меньше 2
49
наименьший натуральный делитель: 7
```

Результат выполнения программы

Задание.

1. Дано вещественное число – цена 1 кг конфет. Вывести стоимость 1, 2, ... 10 кг конфет. Решить задачу используя циклическую конструкцию for.
2. Дана непустая последовательность целых чисел, оканчивающаяся нулем. Найти: а) сумму всех чисел последовательности; б) количество всех чисел последовательности

Решить задачу используя циклическую конструкцию while.

Лабораторная работа 5 Работа со строками в Python

Цель работы: познакомится с методами работы со строками.

Учащийся должен:

Владеть:

Навыками составления линейных алгоритмов на языке программирования Python с использованием строковых данных; **Уметь:**

Применять функции и методы строк при обработке строковых данных;

Знать:

Операции и методы обработки строк.

Строка — базовый тип представляющий из себя неизменяемую последовательность символов; str от «string» — «строка».

Функции и методы работы со строками

Функция или метод	Назначение
S1 + S2	Конкатенация (сложение строк)
S1 * 3	Повторение строки
S[i]	Обращение по индексу
S[i:j:step]	Извлечение среза
len(S)	Длина строки
S.join(список)	Соединение строк из последовательности str через разделитель, заданный строкой
S1.count(S[, i, j])	количество вхождений подстроки s в строку s1. Результатом является число. Можно указать позицию начала поиска i и окончания поиска j

S.find (str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или -1
S.index (str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или вызывает ValueError
S.rindex (str, [start],[end])	Поиск подстроки в строке. Возвращает номер последнего вхождения или вызывает ValueError
S.replace (шаблон, замена)	Замена шаблона
S.split (символ)	Разбиение строки по разделителю
S.upper ()	Преобразование строки к верхнему регистру
S.lower ()	Преобразование строки к нижнему регистру

Ниже приведена программа, демонстрирующая использование функций и методов работы со строками.

```

example_string.py - К:\Лабораторные Python\example_string.py (3.7.1)
File Edit Format Run Options Window Help
s1="Пропаганда"
s2="Сенсация"
s3="Сенсация*Сенсация*Сенсация*Сенсация"
s4='OxOxOxAx'
print('s1 = ',s1)
print('s2 = ',s2)
print('s3 = ',s3)
print('s4 = ',s4)
print('s1+s2 = ',s1+s2) #сложение двух строк
print('s1*3 = ',s1*3) #умножение строки на 3, т.е.строка выведется 3 раза
print('s1[2] = ',s1[2]) #вывод элемента строки s1 с индексом 2
print('s1[2,4] = ',s1[2:4]) #извлечение среза строки s1 начиная с индекса 2
#и заканчивая индексом 4
print('s3.count = ',s3.count(s2)) #количество вхождений подстроки s2 в s3,
#в результате выведется число
print('s1.find('a') = ',s1.find('a')) #поиск подстроки 'a' в строке s1
#результатом будет номер первого вхождения
print('s1.index('п') = ',s1.index('п'))#поиск подстроки 'п' в строке s1
#результатом будет номер первого вхождения
print('s1.rindex('д') = ',s1.rindex('д'))#поиск подстроки 'а' в строке s1
#возвращает номер последнего вхождения
print('s4.replace('Ox','Ax',2) = ',s4.replace('Ox','Ax',2))#замена шаблона. Строка 'Ox' - это шаблон
#строка 'Ax' - это замена
#в строке 4 последовательность 'Ox' будет заменена
#на 'Ax' с шагом 2
print('s3.split('*') = ',s3.split('*'))#разбиение по разделителю *
print('s1.upper = ',s1.upper())#перевод символов в верхний регистр
print('s1.lower = ',s1.lower())#перевод символов в нижний регистр
Ln: 20 Col: 40

```

Пример программы на Python

```

Python 3.7.1 Shell
File Edit Shell Debug Options Window Help
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.191] on win32
Type "help", "copyright", "credits" or "license()" for more
>>>
===== RESTART: К:\Лабораторные Python\example_string.py =====
s1 = Пропаганда
s2 = Сенсация
s3 = Сенсация*Сенсация*Сенсация*Сенсация
s4 = ОхОхОхАх
s1+s2 = ПропагандаСенсация
s1*3 = ПропагандаПропагандаПропаганда
s1[2] = о
s1[2,4] = оп
s3.count = 4
s1.find(а) = 4
s1.index(п) = 3
s1.rindex(д) = 9
s4.replace(Ох,Ах,2) = АхАхОхАх
s3.split(*) = ['Сенсация', 'Сенсация', 'Сенсация', 'Сенсация']
s1.upper = ПРОПАГАНДА
s1.lower = пропаганда

```

Результат выполнения программы с использованием функций и методов работы со строками

Пример

Вариант 0

Проверить, будет ли строка читаться одинаково справа налево и слева направо (т. е. является ли она палиндромом).

Решение

Сначала введём строку командой: `s=input('Введите строку ')`.

Затем определим логическую переменную `flag` и присвоим ей значение `1: flag=1`.

Для начала в введённой строке нужно удалить пробелы. Для этого воспользуемся циклической конструкцией `for`, которая выполнится столько раз, какую имеет длину строка. Длину строки определим функцией `len(s)`.

В теле цикла будем проверять следующее условие: `s[i]!=' '`. Данное логическое выражение будет истинно в том случае, если `i`-ый элемент строки не будет равен пробелу, тогда выполнится команда следующая после двоеточия: `string+=s[i]`.

К строке `string`, которая была объявлена в начале программы, будет добавляться посимвольно строка `s`, но уже без пробелов.

Для проверки строки на "палиндром" воспользуемся циклической конструкцией `for`.

Длина половины строки находится делением нацело на 2. Если количество символов нечетно, то стоящий в середине не учитывается, т.к. его сравниваемая пара - он сам.

Количество повторов цикла равно длине половины строки. Длину строки определим функцией `len(s)`, где аргумент введенная нами строка `s`. Зная длину строки, можно вычислить количество повторов цикла. Для этого целочисленно разделим длину строки на 2: `len(s)//2`.

Для задания диапазона для цикла используем функцию `range()`, в которой аргументом будет являться половина длины строки: `range(len(s)//2)`. `for i in range(len(s)//2)`.

Если символ с индексом `i` не равен "симметричному" символу с конца строки (который находится путем индексации с конца)

`if s[i] != s[-1-i],`

то переменной `flag` присваивается значение 0 и происходит выход из цикла командой `break`.

Далее, при помощи условной конструкции `if-else` в зависимости от значения `flag` либо - 0, либо -1 выводится сообщение, что строка палиндром, либо нет.

```
s=input('Введите строку \n')
flag=1
string=''
for i in range(len(s)):
    if s[i]!=' ':
        string+=s[i]
print(string)
for i in range(len(s)//2):
    if string[i]!=string[-i-1]:
        flag=0
        break
if flag: print('Палиндром')
else: print('не палиндром')
```

Пример программы на Python


```
Введите строку  
а роза упала на лапу азора  
арозаупалана лапуазора  
Палиндром
```

Результат выполнения программы

Задания для самостоятельной работы (по вариантам)

Вариант 1

Дана строка, содержащая русскоязычный текст. Найти количество слов, начинающихся с буквы "е".

Вариант 2

В строке заменить все двоеточия (:) знаком процента (%). Подсчитать количество замен.

Вариант 3

В строке удалить символ точку (.) и подсчитать количество удаленных символов.

Вариант 4

В строке заменить букву(а) буквой (о). Подсчитать количество замен. Подсчитать, сколько символов в строке.

Вариант 5

В строке заменить все заглавные буквы строчными.

Вариант 6

В строке удалить все буквы "а" и подсчитать количество удаленных символов.

Вариант 7

Дана строка. Преобразовать ее, заменив звездочками все буквы "п", встречающиеся среди первых $n/2$ символов. Здесь n - длина строки.

Вариант 8

Дана строка, заканчивающаяся точкой. Подсчитать, сколько слов в строке.

Вариант 9

Определить, сколько раз в тексте встречается заданное слово.

Вариант 10

Дана строка-предложение на английском языке. Преобразовать строку так, чтобы каждое слово начиналось с заглавной буквы. **Вариант 11**

Дана строка. Подсчитать самую длинную последовательность подряд идущих букв «н». Преобразовать ее, заменив точками все восклицательные знаки.

Вариант 12

Дана строка. Вывести все слова, оканчивающиеся на букву "я".

Вариант 13

Дана строка символов, среди которых есть одна открывающаяся и одна закрывающаяся скобки. Вывести на экран все символы, расположенные внутри этих скобок.

Вариант 14

Дана строка. Вывести все слова, начинающиеся на букву "а" и слова оканчивающиеся на букву "я".

Вариант 15

Дана строка текста. Подсчитать количество букв «т» в строке.

Лабораторная работа №16

Работа со списками. Операции над списками в Python

Цель работы: Изучение одномерных массивов в Python.

Массивы (списки) в Python — это определенное количество элементов одного типа, которые имеют общее имя, и каждый элемент имеет свой индекс — порядковый номер.

Часто для работы с массивами используются списки.

Список (list) – это структура данных для хранения объектов различных типов.

Списки являются упорядоченными последовательностями, которые состоят из различных типов данных, заключающихся в квадратные скобки [] и отделяющиеся друг от друга с помощью запятой.

Создание списков на Python.

Создать список можно несколькими способами

1. Получение списка через присваивание конкретных значений.

Так выглядит в коде Python пустой список:

```
s = [] # Пустой список
```

Примеры создания списков со значениями:

```
l=[5, 75, -4, 7, -51] # список целых чисел
l=[1.13, 5.34, 12.63, 4.6, 34.0, 12.8] # список из вещественных чисел
l=["Оля", "Владимир", "Михаил", "Дарья"] # список из строк
l=["Москва", "Иванов", 12, 124] # смешанный список
l=[[0, 0, 0], [1, 0, 1], [1, 1, 0]] # список, состоящий из списков
l=['s', 'p', ['isok'], 2] # список из значений и списка
```

Списки можно складывать (конкатенировать) с помощью знака «+»:

```
l=[1, 3]+[4, 23]+[5]
print('l=[1, 3]+[4, 23]+[5] =', l)
```

Результат:

```
>>>
l=[1, 3]+[4,23]+[5] = [1, 3, 4, 23, 5]
>>> |
```

2. Создание списка при помощи функции Split().

Используя функцию split в Python можно получить из строки

список. stroka = "Привет, страна" lst=stroka.split(",")

```
stroka = "Здравствуй, Дедушка Мороз" #stroka - строка
lst=stroka.split(",") #lst - список
print('stroka = ',stroka)
print('lst=stroka.split(","):',lst)
```

Результат:

```
===== RESTART: C:/Users/maxim/Desktop/ex_list_
stroka = Здравствуй, Дедушка Мороз
lst=stroka.split(","): ['Здравствуй', ' Дедушка Мороз']
```

3. Генераторы списков.

В Python создать список можно также при помощи генераторов.

Первый способ.

Сложение одинаковых списков заменяется

умножением: Список из 10 элементов, заполненный

единицами l = [1]*10

Второй способ.

Пример 1.

```
l = [i for i in range(10)]
```

Пример 2.

```
c = [c * 3 for c in 'list'] print
```

```
(c) # ['lll', 'iii', 'sss', 'ttt']
```

```

Создание списка из строки.
l = list (строка):
 ['c', 'т', 'р', 'о', 'к', 'а']

Создание списка при помощи функции Split().
stroka=" Hello, friend "
lst=stroka.split(","):
 ['Hello', ' friend']

Генераторы списков.
Первый способ.
l = [1]*10:
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

Второй способ. Пример 1.
l = [i for i in range(10)]:
 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Второй способ. Пример 2.
c=[c*3 for c in "list"]:
 ['lll', 'iii', 'sss', 'ttt']

```

Примеры использования генераторов списка.

Пример 1.

Заполнить список квадратами чисел от 0 до 9, используя генератор списка.

Решение:

```
l = [i*i for i in range(10)]
```

Пример 2.

Заполнить список числами, где каждое последующее число больше на 2.

```
l = [(i+1)+i for i in range(10)] print(l)
```

```

Заполнить список квадратами чисел от 0 до 9, используя генератор списка.
l = [i*i for i in range(10)]:
 [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

Заполнить список числами, где каждое последующее число больше на 2.
l = [(i+1)+i for i in range(10)]:
 [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

```

Модуль random предоставляет функции для генерации случайных чисел, букв, случайного выбора элементов последовательности. random.randint(A, B) - случайное целое число N, $A \leq N \leq B$. random.random() - случайное число от 0 до 1.

Случайные числа в списке:

10 чисел, сгенерированных случайным образом в диапазоне (10,80)

```
from random import randint
```

```
l = [randint(10,80) for x in range(10)]
```

10 чисел, сгенерированных случайным образом в диапазоне (0,1)

```
l = [random() for i in range(10)]
```

```
from random import *
l = [randint(10,80) for i in range(10)]
print('10 чисел, сгенерированных случайным образом в диапазоне (10,80).')
print('l = [randint(10,80) for x in range(10)]:')
print(l)
print()
```

```
l = [random() for i in range(10)]
print('10 чисел сгенерированных в диапазоне от 0 до 1.')
print('l = [random() for i in range(10)]:')
for i in range(len(l)):
    print ('{: .2f}'.format(l[i]), end = " ")
```

```
10 чисел, сгенерированных случайным образом в диапазоне (10,80).
l = [randint(10,80) for x in range(10)]:
[70, 33, 79, 61, 34, 27, 11, 55, 52, 31]
```

```
10 чисел сгенерированных в диапазоне от 0 до 1.
l = [random() for i in range(10):
0.66 0.97 0.87 0.57 0.54 0.83 0.57 0.65 0.04 0.07
```

4. Ввод списка (массива) в языке Python.

Для ввода элементов списка используется цикл for и команда range

```
() : for i in range(N): x[i] = int( input() )
```

Более простой вариант ввода списка:

```
x = [ int(input()) for i in range(N) ]
```

```
print('Ввод списка. Пример 1:')
x=[]
for i in range(4):
    x.append(int(input()))
print(x)

x=[]
print('Ввод списка. Пример 2:')
x = [ int(input()) for i in range(4) ]
print(x)
```

Ввод списка. Пример 1:

```
45
4
85
2
[45, 4, 85, 2]
```

Ввод списка. Пример 2:

```
4
5
7
8
[4, 5, 7, 8]
```

Функция `int` здесь используется для того, чтобы строка, введенная пользователем, преобразовывалась в целые числа.

5. Вывод списка (массива) в языке Python.

Вывод целого списка (массива):

```
print (L)
```

Поэлементный вывод списка (массива):

```
for i in range(N):
```

```
    print ( L[i], end = " " )
```

```

'''
Вывод целого списка (массива)
[1, 56, 6, 3, 6, 7, 3, 37, 7, 37, 37]

Поэлементный вывод списка (массива)
1 56 6 3 6 7 3 37 7 37 37
'''
```

2. Методы списков.

Метод	Что делает
list.append(x)	Добавляет элемент в конец списка
list.extend(L)	Расширяет список list, добавляя в конец все элементы списка L
list.insert(i, x)	Вставляет перед i-ым элементом значение x
list.remove(x)	Удаляет первый элемент в списке, имеющий значение x. ValueError, если такого элемента не существует
list.pop([i])	Удаляет i-ый элемент и возвращает его. Если индекс не указан, удаляется последний элемент

list.index(x, [start [, end]])	Возвращает положение первого элемента со значением x (при этом поиск ведется от start до end)
list.count(x)	Возвращает количество элементов со значением x
list.reverse()	Разворачивает список
list.copy()	Поверхностная копия списка
list.clear()	Очищает список

Ниже приведена программа, демонстрирующая методы работы списков.

```

a=[0, 2, 2, 2, 4] #список a
b=[5, 6, 7, 2, 9] #список b
print ('Исходный список a:', a)
print ('Исходный список b:', b)
x=99
y=5

a.append(x)
print ('a.append(x):', a)

a.extend(b)
print ('a.extend(b):', a)

a.insert(3, x)
print ('a.insert(3, x):', a)

a.remove(x)
print ('a.remove(x):', a)

print ('a.pop(5):', a.pop(5))
print (a)

print ('a.index(y, 0, len(a)):', a.index(y, 0, len(a)))

print ('a.count(2):', a.count(2))

a.reverse()
print ('a.reverse():', a)

z=a.copy()
print ('z=a.copy():', z)

z.clear()
print ('z.clear():')
print ('z =', z)

```

Пример программы на Python


```

Исходный список a: [0, 2, 2, 2, 4]
Исходный список b: [5, 6, 7, 2, 9]
a.append(x): [0, 2, 2, 2, 4, 99]
a.extend(b): [0, 2, 2, 2, 4, 99, 5, 6, 7, 2, 9]
a.insert(3,x): [0, 2, 2, 99, 2, 4, 99, 5, 6, 7, 2, 9]
a.remove(x): [0, 2, 2, 2, 4, 99, 5, 6, 7, 2, 9]
a.pop(5): 99
[0, 2, 2, 2, 4, 5, 6, 7, 2, 9]
a.index(y,0,len(a)): 5
a.count(2): 4
a.reverse(): [9, 2, 7, 6, 5, 4, 2, 2, 2, 0]
z=a.copy(): [9, 2, 7, 6, 5, 4, 2, 2, 2, 0]
z.clear():
z = []

```

Результат выполнения программы

Вариант 0

1. Из массива X длиной n, среди элементов которого есть положительные, отрицательные и равные нулю, сформировать новый массив Y, взяв в него только те элементы из X, которые больше по модулю заданного числа M. Вывести на экран число M, данный и полученные массивы.

Решение:

```

n=int(input('Введите длину массива\n'))
m=int(input('Введите число M\n'))
x=[]
y=[]
for i in range(n):
    print('Введите ',i,'элемент:')
    x.append(int(input()))
for i in range(n):
    if abs(x[i])>m:
        y.append(x[i])
print('Введённое число M:',m)
print('Массив X:',x)
print('Массив Y:',y)

```

```

Введите длину массива
5
Введите число M
20
Введите 0 элемент:
21
Введите 1 элемент:
22
Введите 2 элемент:
5
Введите 3 элемент:
6
Введите 4 элемент:
8
Введённое число M: 20
Массив X: [21, 22, 5, 6, 8]
Массив Y: [21, 22]

```

2. В массиве целых чисел все отрицательные элементы заменить на положительные. Вывести исходный массив и полученный.

Решение:

```

n=int(input('Введите длину массива:'))
a=[]
for i in range(n):
    print('Введите',i,'элемент:')
    a.append(int(input()))
print('Исходный массив:',a)
for i in range(n):
    if a[i]<0:
        a[i]=-a[i]
print('Полученный массив:',a)

```

```

Введите длину массива:5
Введите 0 элемент:
-5
Введите 1 элемент:
-4
Введите 2 элемент:
-6
Введите 3 элемент:
5
Введите 4 элемент:
-7
Исходный массив: [-5, -4, -6, 5, -7]
Полученный массив: [5, 4, 6, 5, 7]

```

Вариант 1

1. Дан одномерный массив, состоящий из N целочисленных элементов. Ввести массив с клавиатуры. Найти максимальный элемент. Вывести массив на экран в обратном порядке.
2. В массиве действительных чисел все нулевые элементы заменить на среднее арифметическое всех элементов массива.

Вариант 2

1. Дан одномерный массив, состоящий из N целочисленных элементов. Ввести массив с клавиатуры. Найти минимальный элемент. Вывести индекс минимального элемента на экран.
2. Дан массив целых чисел. Переписать все положительные элементы во второй массив, а остальные - в третий.

Вариант 3

1. В одномерном числовом массиве D длиной n вычислить сумму элементов с нечетными индексами. Вывести на экран массив D , полученную сумму.
2. Дан одномерный массив из 8 элементов. Заменить все элементы массива меньшие 15 их удвоенными значениями. Вывести на экран монитора преобразованный массив.

Вариант 4

1. Дан массив целых чисел. Найти максимальный элемент массива и его порядковый номер.
2. Дан одномерный массив целого типа. Получить другой массив, состоящий только из нечетных чисел исходного массива или сообщить, что таких чисел нет. Полученный массив вывести в порядке убывания элементов.

Вариант 5

1. Дан одномерный массив из 10 целых чисел. Вывести пары отрицательных чисел, стоящих рядом.
2. Дан целочисленный массив размера 10. Создать новый массив, удалив все одинаковые элементы, оставив их 1 раз.

Вариант 6

1. Дан одномерный массив из 10 целых чисел. Найти максимальный элемент и сравнить с ним остальные элементы. Вывести количество меньших максимального и больших максимального элемента.
2. Одномерный массив из 10-и целых чисел заполнить с клавиатуры, определить сумму тех чисел, которые >5 .

Вариант 7

1. Дан массив целых чисел. Найти сумму элементов с четными номерами и произведение элементов с нечетными номерами. Вывести сумму и произведение.
2. Переставить в одномерном массиве минимальный элемент и максимальный.

Вариант 8

1. Найдите сумму и произведение элементов списка. Результаты вывести на экран.
2. В массиве действительных чисел все нулевые элементы заменить на среднее арифметическое всех элементов массива.

Вариант 9

1. Дан одномерный массив, состоящий из N вещественных элементов. Ввести массив с клавиатуры. Найти и вывести минимальный по модулю элемент.
Вывести массив на экран в обратном порядке.
2. Даны массивы A и B одинакового размера 10. Вывести исходные массивы. Поменять местами их содержимое и вывести в начале элементы преобразованного массива A , а затем — элементы преобразованного массива B .

Вариант 10

1. Определите, есть ли в списке повторяющиеся элементы, если да, то вывести на экран это значение, иначе сообщение об их отсутствии.
2. Дан одномерный массив из 15 элементов. Элементам массива меньше 10 присвоить нулевые значения, а элементам больше 20 присвоить 1. Вывести на экран монитора первоначальный и преобразованный массивы в строчку.

Вариант 11

1. Найти наибольший элемент списка, который делится на 2 без остатка и вывести его на экран.
2. Дан одномерный массив целого типа. Получить другой массив, состоящий только из четных чисел исходного массива, меньше 10, или сообщить, что таких чисел нет. Полученный массив вывести в порядке возрастания элементов.

Вариант 12

1. Найти наименьший нечетный элемент списка и вывести его на экран.
2. Даны массивы А и В одинакового размера 10. Поменять местами их содержимое и вывести вначале элементы преобразованного массива А, а затем — элементы преобразованного массива В.

Вариант 13

1. Дан одномерный массив целых чисел. Проверить, есть ли в нем одинаковые элементы. Вывести эти элементы и их индексы.
2. Дан одномерный массив из 8 элементов. Заменить все элементы массива меньшие 15 их удвоенными значениями. Вывести на экран монитора преобразованный массив.

Вариант 14

1. Найти максимальный элемент численного массива и поменять его местами с минимальным.
2. Программа заполняет одномерный массив из 10 целых чисел числами, считанными с клавиатуры. Определить среднее арифметическое всех чисел массива. Заменить элементы массива большие среднего арифметического на 1.

Вариант 15

1. Определите, есть ли в списке повторяющиеся элементы, если да, то вывести на экран эти значения.
2. Дан одномерный массив целого типа. Получить другой массив, состоящий только из нечетных чисел исходного массива или сообщить, что таких чисел нет. Полученный массив вывести в порядке убывания элементов.

Лабораторная работа №17 Функции и процедуры в Python

Цель работы: изучение процедур и функций в Python.

знать - синтаксис процедур и функций, процедура с параметром, локальные и глобальные переменные;

уметь - применять синтаксис процедур и функций при составлении программы;

владеть - основными навыками работы с функциями и процедурами.

Подпрограмма - это именованный фрагмент программы, к которому можно обратиться из другого места программы

Подпрограммы делятся на две категории: процедуры и функции.

1. Процедуры.

Рассмотрим синтаксис процедуры:

```
def имя процедуры(Список параметров):
```

```
    Система команд
```

Для определения процедуры используется ключевое слово `def`, затем указывается имя процедуры и в скобках её формальные параметры, если они присутствуют. После ставится двоеточие и со следующей строки с отступом в 4 пробела указываются команды.

Процедура — вспомогательный алгоритм, выполняющий некоторые действия. Процедура должна быть определена к моменту её вызова. Определение процедуры начинается со служебного слова `def`.

Вызов процедуры осуществляется по ее имени, за которым следуют круглые скобки, например, `Err()`.

В одной программе может быть сколько угодно много вызовов одной и той же процедуры.

Использование процедур сокращает код и повышает удобочитаемость.

Процедура с параметрами.

Как используются в Python параметры процедуры, рассмотрим на примере.

Пример.

Написать процедуру, которая печатает раз указанный символ (введенный с клавиатуры), каждый с новой строки.

```
def printChar(s):
    print (s)
sim = input('введите символ')
printChar(sim) # первый вызов, вывод введенного символа
printChar('*') # второй вызов, вывод *
```

```
def printChar(s):
    print (s)
sim = input('введите символ: |')
printChar(sim) # первый вызов, вывод введенного символа
printChar('*') # второй вызов, вывод *

>>>
введите символ: 41
41
*
```

Глобальная переменная — если ей присвоено значение в основной программе (вне процедуры).

Локальная переменная (внутренняя) известна только на уровне процедуры, обратиться к ней из основной программы и из других процедур нельзя.

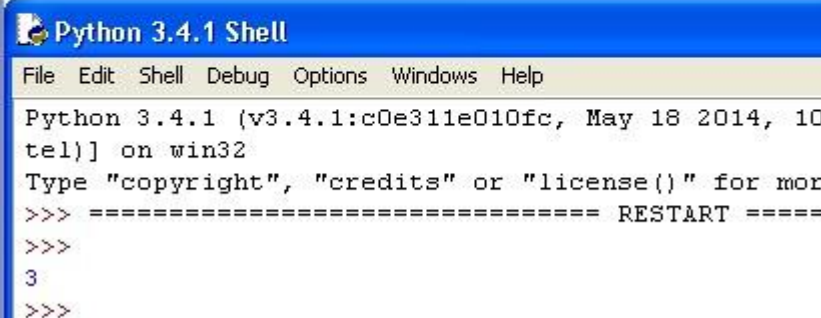
Параметры процедуры — локальные переменные.

2. Примеры использования локальных и глобальных переменных.

Пример 1.

```
x = 3 # глобальная переменная def
pr(): # процедура без параметров
    print (x) # вывод значения глобальной переменной
pr()
```

```
x = 3 # глобальная переменная
def pr(): # процедура без параметров
    print (x) # вывод значения глобальной переменной
pr()
```



Python 3.4.1 Shell

File Edit Shell Debug Options Windows Help

Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:30:14) on win32
Type "copyright", "credits" or "license()" for more
>>> ===== RESTART =====
>>>
3
>>>

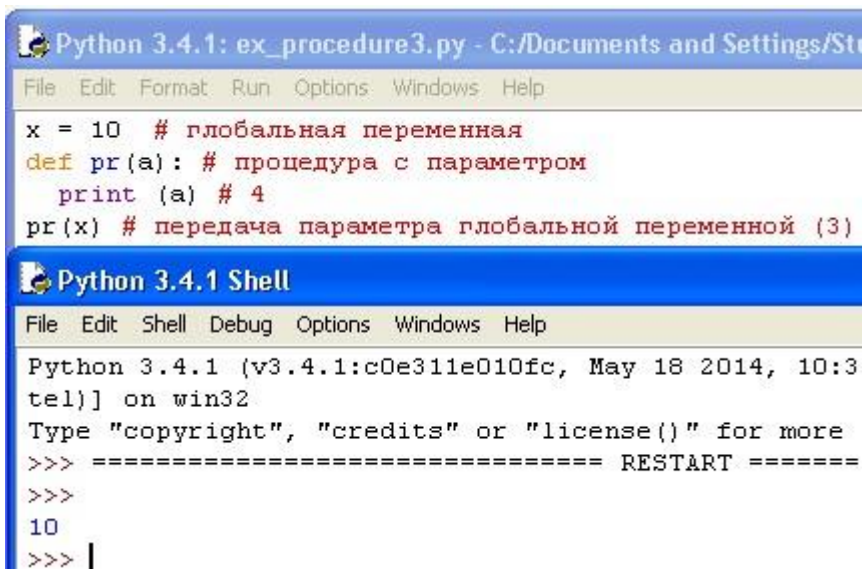
Пример 2.

```
x = 3 # глобальная переменная def
```

```
pr(a): # процедура с параметром
```

```
    print (a) # 4
```

```
pr(x) # передача параметра глобальной переменной (3)
```



Python 3.4.1: ex_procedure3.py - C:/Documents and Settings/St...

File Edit Format Run Options Windows Help

```
x = 10 # глобальная переменная
def pr(a): # процедура с параметром
    print (a) # 4
pr(x) # передача параметра глобальной переменной (3)
```

Python 3.4.1 Shell

File Edit Shell Debug Options Windows Help

Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:30:14) on win32
Type "copyright", "credits" or "license()" for more
>>> ===== RESTART =====
>>>
10
>>> |

Существует возможность изменить значение глобальной переменной (не создавая локальную). В процедуре с помощью слова `global`:

```
x = 3 # глобальная переменная
```



```
def pr(): # процедура без
параметров  global x  x =
pow(x,10)
```

```
    print (x) # вывод измененного значения глобальной переменной
pr()
```

The screenshot shows two windows from a Python 3.4.1 IDE. The top window, titled 'Python 3.4.1: ex_procedure4.py - C:\Docume', displays the following code:

```
x=3 # глобальная переменная
print ('Начальное значение: ',x)
def pr(): # процедура без параметров
    global x
    x=pow(x,10)
    print ('Изменённое значение: ',x)
pr()
```

The bottom window, titled 'Python 3.4.1 Shell', shows the execution output:

```
Python 3.4.1 (v3.4.1:c0e311e010fc, May
tel)] on win32
Type "copyright", "credits" or "licens
>>> ===== I
>>>
Начальное значение:  3
Изменённое значение:  59049
>>>
```

3. Функции.

Функция - подпрограмма, к которому можно обратиться из другого места программы. Для создания функции используется ключевое слово `def`, после которого указывается имя и список аргументов в круглых скобках. Тело функции выделяется также как тело условия (или цикла): четырьмя пробелами.

Рассмотрим синтаксис функции:

```
def имя функции(Список
параметров): Система команд
return выражение
```

Часть функций языка Python являются встроенными функциями, которые обеспечены синтаксисом самого языка. Например, `int`, `input`, `randint`. Рассмотрим пример создания пользовательских функций.

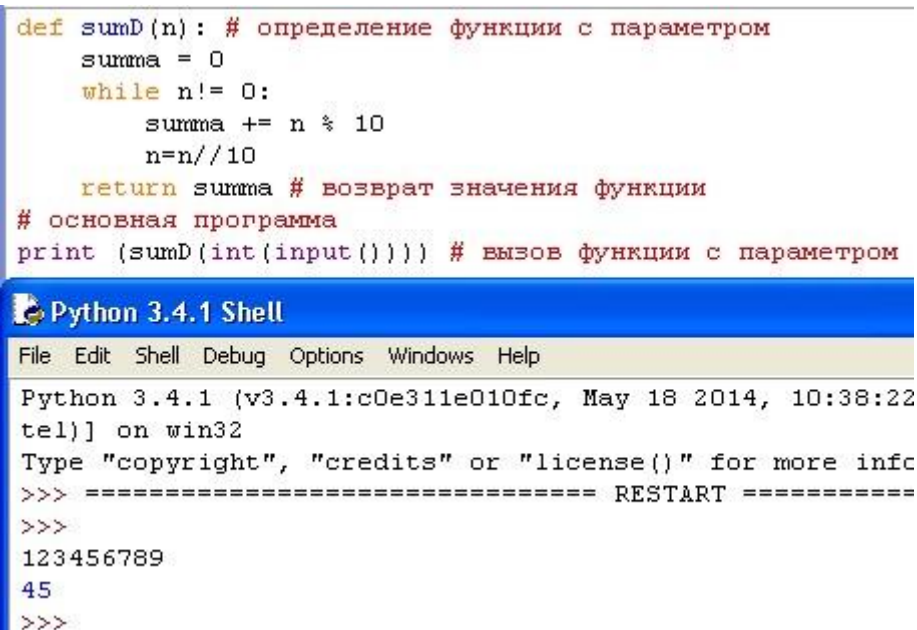
Пример 1.

```

Вычислить сумму цифр числа. def sumD(n): #
определение функции с параметром    sumD = 0
while n!= 0:

    sumD += n % 10
    n = n // 10
return sumD # возврат значения функции
# основная программа
print (sumD(int(input()))) # вызов функции с параметром

```



```

def sumD(n): # определение функции с параметром
    summa = 0
    while n!= 0:
        summa += n % 10
        n=n//10
    return summa # возврат значения функции
# основная программа
print (sumD(int(input()))) # вызов функции с параметром

```

Python 3.4.1 Shell

File Edit Shell Debug Options Windows Help

Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:38:22
tel)] on win32
Type "copyright", "credits" or "license()" for more info
>>> ===== RESTART =====
>>>
123456789
45
>>>

Вариант 0.

1. Определить, являются ли три треугольника равновеликими. Длины сторон вводятся с клавиатуры. Для подсчёта площади треугольника использовать формулу Герона. Вычисление площади оформить в виде функции с тремя параметрами.

Формула Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

$$\text{где } p = \frac{a+b+c}{2}$$

Решение:

```
import math
def s(x, y, z):
    p=(x+y+z)/2
    s=math.sqrt(p*(p-x)*(p-y)*(p-z))
    return s
A=[]
for i in range(3):
    print('Введите стороны ',i,'-го треугольника:')
    a=int(input('a:'))
    b=int(input('b:'))
    c=int(input('c:'))
    A.append(s(a,b,c))
for i in range(3):
    print('Площадь ',i,'-го треугольника {:.2f}'.format(A[i]))
if A[0]==A[1]:
    if A[0]==A[2]:
        print('Треугольники равновеликие')
else: print('Треугольники не равновеликие')
```

```
Введите стороны 0 -го треугольника:
a:3
b:4
c:5
Введите стороны 1 -го треугольника:
a:6
b:7
c:8
Введите стороны 2 -го треугольника:
a:9
b:10
c:11
Площадь 0 -го треугольника 6.00
Площадь 1 -го треугольника 20.33
Площадь 2 -го треугольника 42.43
Треугольники не равновеликие
```

2. Ввести одномерный массив А длиной m. Поменять в нём местами первый и последний элементы. Длину массива и его элементы ввести с клавиатуры. В программе описать процедуру для замены элементов массива. Вывести исходные и полученные массивы.

Решение:

```
def zam(X):
    tmp=X[0]
    X[0]=X[len(X)-1]
    X[len(X)-1]=tmp
A=[]
m=int(input('Введите длину массива:'))
for i in range(m):
    print('Введите ',i,'элемент массива')
    A.append(int(input()))
print(A)
zam(A)
print(A)
```

```
Введите длину массива:5
Введите 0 элемент массива
0
Введите 1 элемент массива
1
Введите 2 элемент массива
2
Введите 3 элемент массива
3
Введите 4 элемент массива
4
[0, 1, 2, 3, 4]
[4, 1, 2, 3, 0]
```

Вариант 1.

1. Составить программу для вычисления площади разных геометрических фигур.
2. Даны 3 различных массива целых чисел (размер каждого не превышает 15). В каждом массиве найти сумму элементов и среднеарифметическое значение. **Вариант 2.**

1. Вычислить площадь правильного шестиугольника со стороной a , используя подпрограмму вычисления площади треугольника.
2. Пользователь вводит две стороны трех прямоугольников. Вывести их площади.

Вариант 3.

1. Даны катеты двух прямоугольных треугольников. Написать функцию вычисления длины гипотенузы этих треугольников. Сравнить и вывести какая из гипотенуз больше, а какая меньше.
2. Преобразовать строку так, чтобы буквы каждого слова в ней были отсортированы по алфавиту.

Вариант 4.

1. Даны две дроби A/B и C/D (A, B, C, D — натуральные числа). Составить программу деления дроби на дробь. Ответ должен быть несократимой дробью. Использовать подпрограмму алгоритма Евклида для определения НОД.
2. Задана окружность $(x-a)^2 + (y-b)^2 = R^2$ и точки $P(p_1, p_2)$, $F(f_1, f_1)$, $L(l_1, l_2)$. Выяснить и вывести на экран, сколько точек лежит внутри окружности. Проверку, лежит ли точка внутри окружности, оформить в виде процедуры.

Вариант 5.

1. Даны две дроби A/B и C/D (A, B, C, D — натуральные числа). Составить программу вычитания из первой дроби второй. Ответ должен быть несократимой дробью. Использовать подпрограмму алгоритма Евклида для определения НОД.
2. Напишите программу, которая выводит в одну строчку все делители переданного ей числа, разделяя их пробелами.

Вариант 6.

1. Составить программу нахождения наибольшего общего делителя (НОД) и наименьшего общего кратного (НОК) двух натуральных чисел $\text{НОК}(A, B) =$

$(A*B)/НОД(A,B)$. Использовать подпрограмму алгоритма Евклида для определения НОД.

2. Составить программу вычисления площади выпуклого четырехугольника, заданного длинами четырех сторон и диагонали.

Вариант 7.

1. Даны числа X, Y, Z, T — длины сторон четырехугольника. Вычислить его площадь, если угол между сторонами длиной X и Y — прямой. Использовать две подпрограммы для вычисления площадей: прямоугольного треугольника и прямоугольника.

2. Напишите программу, которая переводит переданное ей неотрицательное целое число в 10-значный восьмеричный код, сохранив лидирующие нули.

Вариант 8.

1. Найти все натуральные числа, не превосходящие заданного n , которые делятся на каждую из своих цифр.

2. Ввести одномерный массив A длиной m . Поменять в нём местами первый и последний элементы. Длину массива и его элементы ввести с клавиатуры. В программе описать процедуру для замены элементов массива. Вывести исходные и полученные массивы.

Вариант 9.

1. Из заданного числа вычли сумму его цифр. Из результата вновь вычли сумму его цифр и т. д. Через сколько таких действий получится нуль?

2. Даны 3 различных массива целых чисел. В каждом массиве найти произведение элементов и среднеарифметическое значение.

Вариант 10.

1. На отрезке $[100, N]$ ($210 < N < 231$) найти количество чисел, составленных из цифр a, b, c .

2. Составить программу, которая изменяет последовательность слов в строке на обратную.

Вариант 11.

1. Два простых числа называются «близнецами», если они отличаются друг от друга на 2 (например, 41 и 43). Напечатать все пары «близнецов» из отрезка $[n, 2n]$, где n — заданное натуральное число, большее 2.
2. Даны две матрицы A и B . Написать программу, меняющую местами максимальные элементы этих матриц. Нахождение максимального элемента матрицы оформить в виде процедуры.

Вариант 12.

1. Два натуральных числа называются «дружественными», если каждое из них равно сумме всех делителей (кроме его самого) другого (например, числа 220 и 284). Найти все пары «дружественных» чисел, которые не больше данного числа N .
2. Даны длины сторон треугольника a, b, c . Найти медианы треугольника, сторонами которого являются медианы исходного треугольника. Для вычисления медианы проведенной к стороне a , использовать формулу. Вычисление медианы оформить в виде процедуры.

Вариант 13.

1. Натуральное число, в записи которого n цифр, называется числом Армстронга, если сумма его цифр, возведенная в степень n , равна самому числу. Найти все числа Армстронга от 1 до k .
2. Три точки заданы своими координатами $X(x_1, x_2)$, $Y(y_1, y_2)$ и $Z(z_1, z_2)$. Найти и напечатать координаты точки, для которой угол между осью абсцисс и лучом, соединяющим начало координат с точкой, минимальный. Вычисления оформить в виде процедуры.

Вариант 14.

1. Составить программу для нахождения чисел из интервала $[M, N]$, имеющих наибольшее количество делителей.
2. Четыре точки заданы своими координатами $X(x_1, x_2)$, $Y(y_1, y_2)$, $Z(z_1, z_2)$, $P(p_1, p_2)$. Выяснить, какие из них находятся на максимальном расстоянии друг от друга и вывести на экран значение этого расстояния. Вычисление расстояния между двумя точками оформить в виде процедуры.

Вариант 15.

1. Найти все простые натуральные числа, не превосходящие n , двоичная запись которых представляет собой палиндром, т. е. читается одинаково слева направо и справа налево.
2. Четыре точки заданы своими координатами $X(x_1, x_2, x_3)$, $Y(y_1, y_2, y_3)$, $Z(z_1, z_2, z_3)$, $T(t_1, t_2, t_3)$. Выяснить, какие из них находятся на минимальном расстоянии друг от друга и вывести на экран значение этого расстояния. Вычисление расстояния между двумя точками оформить в виде процедуры.

Лабораторная работа 8 Работа с двумерными массивами.

Цель работы: изучение двумерных массивов в Python.

знать - способ описания двумерного массива, способы ввода элементов двумерного массива;

уметь - вводить массивы, получать списки через присваивание конкретных значений, применять функции;

владеть - основными навыками создания программ обработки двумерных массивов.

Матрицами называются массивы элементов, представленные в виде прямоугольных таблиц, для которых определены правила математических действий. Элементами матрицы могут являться числа, алгебраические символы или математические функции.

Для работы с матрицами в Python также используются списки. Каждый элемент списка-матрицы содержит вложенный список.

Таким образом, получается структура из вложенных списков, количество которых определяет количество столбцов матрицы, а число элементов внутри каждого вложенного списка указывает на количество строк в исходной матрице.

1. Создание списка

Пусть даны два числа: количество строк n и количество столбцов m . Необходимо создать список размером $n \times m$, заполненный нулями. Очевидное решение оказывается неверным:

$$A = [[0] * m] * n$$

В этом легко убедиться, если присвоить элементу $A[0][0]$ значение 1, а потом вывести значение другого элемента $A[1][0]$ — оно тоже будет равно 1! Дело в том, что $[0] * m$ возвращает ссылку на список из m нулей. Но последующее повторение этого элемента создает список из n элементов, которые являются ссылкой на один и тот же список (точно так же, как выполнение операции $B = A$ для списков не создает новый список), поэтому все строки результирующего списка на самом деле являются одной и той же строкой.

Таким образом, двумерный список нельзя создавать при помощи операции повторения одной строки.

Первый способ.

Сначала создадим список из n элементов (для начала просто из n нулей). Затем сделаем каждый элемент списка ссылкой на другой одномерный список из m элементов:

```
A = [0] * n
for i in range(n):
    A[i] = [0] * m
```

```
n=3
m=3
A=[0]*n
for i in range(n):
    A[i]=[0]*m
print('A:',A)
```

```
A: [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
>>> |
```

Второй способ.

Создать пустой список, потом n раз добавить в него новый элемент, являющийся списком-строкой:

```
A = []
for i in range(n):
    A.append([0] * m)
```

```
n=3
m=4
A = []
for i in range(n):
    A.append([0] * m)
print(A)
```

```
A: [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
>>> |
```

2. Ввод вложенного списка (двумерного массива)

Пример:

```
n=5
A = []
for i in range(n):
    row = input()
    for i in range(len(row)):
        row[i] = int(row[i])
    A.append(row)
```

The image shows two windows from a Python IDE. The top window, titled 'ex_array_primer.py - E:/ex_array_primer.py (3.4.3)', contains the following Python code:

```
n=3
A = []
for i in range(n):
    B = []
    for i in range(n):
        B.append(int(input()))
    A.append(B)
```

The bottom window, titled 'Python 3.4.3 Shell', shows the execution output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601,
tel)] on win32
Type "copyright", "credits" or "l:
>>> =====
>>>
1
2
3
4
5
6
7
8
9
>>> |
```

3. Вывод вложенного списка (двумерного массива)

Для обработки и вывода списка как правило используется два вложенных цикла. Первый цикл по номеру строки, второй цикл по элементам внутри строки. Например, вывести двумерный числовой список на экран построчно, разделяя числа пробелами внутри одной строки, можно так:

```
for i in range(n):
    for j in range(n):
        print(A[i][j], end = ' ')
    print()
```

```

n=3
A = []
#ввод массива
for i in range(n):
    B = []
    for i in range(n):
        B.append(int(input()))
    A.append(B)
#вывод массива
for i in range(n):
    for j in range(n):
        print(A[i][j], end=' ')
    print()

```

```

1
2
3
4
5
6
7
8
9
1 2 3
4 5 6
7 8 9

```

То же самое, но циклы не по индексу, а по значениям списка:

```

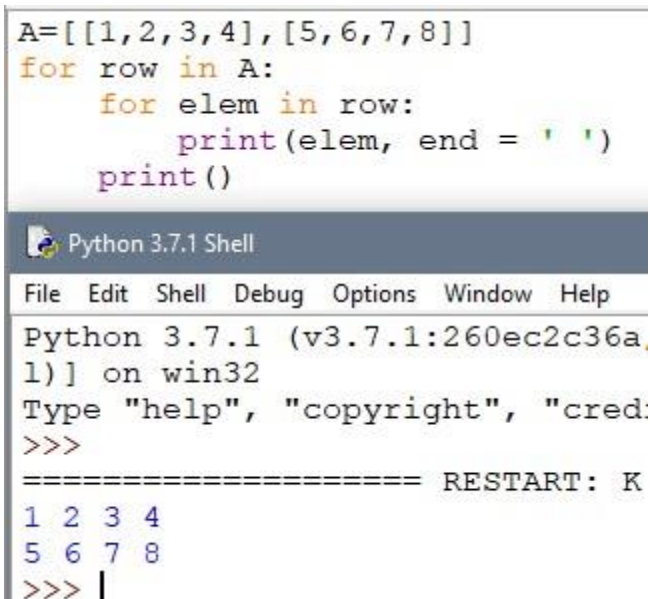
for row in A:
    for elem in row:
        print(elem, end = ' ')
print()

```

```

A=[[1,2,3,4],[5,6,7,8]]
for row in A:
    for elem in row:
        print(elem, end = ' ')
    print()

```



```

Python 3.7.1 Shell
File Edit Shell Debug Options Window Help
Python 3.7.1 (v3.7.1:260ec2c36a,
1)] on win32
Type "help", "copyright", "cred:
>>>
===== RESTART: K
1 2 3 4
5 6 7 8
>>> |

```

Для вывода одной строки можно воспользоваться методом `join`. Используя этот метод в цикле `for` можно `for row in A: print(' '.join(list(map(str, row))))`

4. Обработка и вывод вложенных списков

Часто в задачах приходится хранить прямоугольные таблицы с данными. Такие таблицы называются матрицами или двумерными массивами. В языке программирования Python таблицу можно представить в виде списка строк, каждый элемент которого является в свою очередь списком, например, чисел. Например, создать числовую таблицу из двух строк и трех столбцов можно так:

```
A = [ [1, 2, 3], [4, 5, 6] ]
```

Здесь первая строка списка `A[0]` является списком из чисел `[1, 2, 3]`.

То есть

```
A[0][0]= 1,
```

```
A[0][1]= 2,
```

```
A[0][2]= 3,
```

```
A[1][0]=4,
```

```
A[1][1]=5,
A[1][2]=6.
```

Используем два вложенных цикла для подсчета суммы всех чисел в списке:

```
S = 0
for i in range(len(A)):
    for j in range(len(A[i])):
        S += A[i][j]
```

Или то же самое с циклом не по индексу, а по значениям строк:

```
S = 0
for row in A:
    for elem in row:
        S += elem
```

```
A=[[1, 2, 3,4],[ 5, 6,7,8]]
#вывод при помощи цикла for и метода join
print('Массив A:')
for i in A:
    print(' '.join(list(map(str, i))))
#Пример 1. Подсчёт суммы всех элементов
s = 0
for i in range(len(A)):
    for j in range(len(A[i])):
        s += A[i][j]
print('Пример 1. Сумма элементов:', s)
#Пример 2. Подсчёт суммы всех элементов
s = 0
for row in A:
    for elem in row:
        s += elem
print('Пример 2. Сумма элементов:', s)
```

```
Массив A:
1 2 3 4
5 6 7 8
Пример 1. Сумма элементов: 36
Пример 2. Сумма элементов: 36
```

5. Пример сложной обработки массива

Пусть дана квадратная матрица из n строк и n столбцов. Необходимо элементам, находящимся на главной диагонали, проходящей из левого верхнего угла в правый нижний (то есть тем элементам $A[i][j]$, для которых $i=j$) присвоить значение 0, элементам, находящимся выше главной диагонали – значение 1, элементам, находящимся ниже главной диагонали – значение 2. То есть получить такой массив (пример для $n=3$):

```
0 1 1
2 0 1
2 2 0
```

Рассмотрим несколько способов решения этой задачи.

Первый способ.

Элементы, которые лежат выше главной диагонали – это элементы $A[i][j]$, для которых $i < j$, а для элементов ниже главной диагонали $i > j$. Таким образом, мы можем сравнивать значения i и j и по ним определять значение $A[i][j]$. Получаем следующий алгоритм:

```
for i in range(n):
    for j in range(n):
        if i < j:
            A[i][j] = 0
        elif i > j:
            A[i][j] = 2
        else:
            A[i][j] = 1
```

Ниже приведён пример программы, в котором квадратная матрица 3×3 заполняется элементами со значением 0, а затем ЭЛЕМЕНТАМ, находящимся на главной диагонали, проходящей из левого верхнего угла в правый нижний (то есть тем элементам $A[i][j]$, для которых $i=j$) присваивается значение 0, элементам, находящимся выше главной

диагонали – значение 1, элементам, находящимся ниже главной диагонали – значение 2.

```
n=3
A=[]
#заполняем массив 9-ми
for i in range(n):
    A.append([9]*n)
#вывод исходного массива
for i in range(n):
    for j in range(n):
        print(A[i][j], end = ' ')
    print()
#заменяем элементы главной диагонали, выше и ниже неё
for i in range(n):
    for j in range(n):
        if i < j:
            A[i][j] = 1
        elif i > j:
            A[i][j] = 2
        else:
            A[i][j] = 0
#вывод изменённого массива
print()
for i in range(n):
    for j in range(n):
        print(A[i][j], end = ' ')
    print()
```

```
9 9 9
9 9 9
9 9 9

0 1 1
2 0 1
2 2 0
>>> |
```

Второй способ.

Данный алгоритм плох, поскольку выполняет одну или две инструкции if для обработки каждого элемента. Если мы усложним алгоритм, то мы сможем обойтись вообще без условных инструкций.

Сначала заполним главную диагональ, для чего нам понадобится один цикл:

```
for i in range(n):
    A[i][i] = 1
```

Затем заполним значением 0 все элементы выше главной диагонали, для чего нам понадобится в каждой из строк с номером i присвоить значение элементам $A[i][j]$ для $j=i+1, \dots, n-1$. Здесь нам понадобятся вложенные циклы:

```
for i in range(n):
    for j
    in range(i + 1, n):
        A[i][j] = 0
```

Аналогично присваиваем значение 2 элементам $A[i][j]$ для $j=0, \dots, i-1$:

```
for i in range(n):
    for j in range(0, i):
        A[i][j] = 2
```

Можно также внешние циклы объединить в один и получить еще одно, более компактное решение:

```
for i in range(n):
    for j in range(0, i):
        A[i][j] = 2
    A[i][i]
    = 1
    for j in range(i +
    1, n):
        A[i][j] = 0
```

```

n=3
A=[]
#заполняем массив 9-ми
for i in range(n):
    A.append([9]*n)
#вывод исходного массива
for i in range(n):
    for j in range(n):
        print(A[i][j], end = ' ')
    print()
#заменяем элементы главной диагонали, выше и ниже неё
for i in range(n):
    for j in range(0, i):
        A[i][j] = 2
    A[i][i] = 0
    for j in range(i + 1, n):
        A[i][j] = 1
#вывод изменённого массива
print()
for i in range(n):
    for j in range(n):
        print(A[i][j], end = ' ')
    print()

```

```

9 9 9
9 9 9
9 9 9

0 1 1
2 0 1
2 2 0
>>> |

```

Третий способ.

А вот такое решение использует операцию повторения списков для построения очередной строки списка. i -я строка списка состоит из i чисел 2, затем идет одно число 1, затем идет $n-i-1$ число 0:

```

for i in range(n):
    A[i] = [2] * i + [1] + [0] * (n - i - 1)

```

```

n=3
A=[]
#заполняем массив 9-ми
for i in range(n):
    A.append([9]*n)
#вывод исходного массива
for i in range(n):
    for j in range(n):
        print(A[i][j], end = ' ')
    print()
#заменяем элементы главной диагонали, выше и ниже неё
for i in range(n):
    A[i] = [2] * i + [0] + [1] * (n - i - 1)
#вывод изменённого массива
print()
for i in range(n):
    for j in range(n):
        print(A[i][j], end = ' ')
    print()

```

```

9 9 9
9 9 9
9 9 9

0 1 1
2 0 1
2 2 0
>>> |

```

Вариант 0

1. Дан двумерный массив размером 3×3 . Определить максимальное значение среди элементов третьего столбца массива; максимальное значение среди элементов второй строки массива. Вывести полученные значения.

Решение:

```
n=3
a=[]
for i in range(n):
    b = []
    for j in range(n):
        print('Введите [' , i , ' , ' , j , ' ] элемент')
        b.append(int(input()))
    a.append(b)
#вывод массива
for i in range(n):
    for j in range(n):
        print(a[i][j], end=' ')
    print()

#максимальное значение среди элементов третьего столбца
maximum=a[0][2]
for i in range(n):
    for j in range(n):
        if maximum<a[i][2]:
            maximum=a[i][2]
print('Максимальный в 3 столбце:', maximum)

#максимальное значение среди элементов второй строки
maximum=a[1][0]
for i in range(n):
    for j in range(n):
        if maximum<a[1][j]:
            maximum=a[1][j]
print('Максимальный во второй строке:', maximum)
```

```
Введите [ 0 , 0 ] элемент
1
Введите [ 0 , 1 ] элемент
2
Введите [ 0 , 2 ] элемент
3
Введите [ 1 , 0 ] элемент
4
Введите [ 1 , 1 ] элемент
5
Введите [ 1 , 2 ] элемент
6
Введите [ 2 , 0 ] элемент
7
Введите [ 2 , 1 ] элемент
8
Введите [ 2 , 2 ] элемент
9
1 2 3
4 5 6
7 8 9
Максимальный в 3 столбце: 9
Максимальный во второй строке: 6
```

2. Дан двумерный массив размером $m \times n$.
Сформировать новый массив заменив положительные элементы единицами, а отрицательные нулями.
Вывести оба массива.

Решение:

```
m=int(input('Введите количество строк'))
n=int(input('Введите количество столбцов'))
a=[]
for i in range(m):
    b = []
    for j in range(n):
        print('Введите [' ,i, ',' ,j, '] элемент')
        b.append(int(input()))
    a.append(b)
#Вывод массива
print('Исходный массив:')
for i in range(m):
    for j in range(n):
        print(a[i][j], end=' ')
    print()

for i in range(m):
    for j in range(n):
        if a[i][j]<0: a[i][j]=0
        elif a[i][j]>0: a[i][j]=1

#Вывод массива
print('Изменённый массив:')
for i in range(m):
    for j in range(n):
        print(a[i][j], end=' ')
    print()
```

```

Введите количество строк:3
Введите количество столбцов:4
Введите [ 0 , 0 ] элемент
-1
Введите [ 0 , 1 ] элемент
5
Введите [ 0 , 2 ] элемент
4
Введите [ 0 , 3 ] элемент
-5
Введите [ 1 , 0 ] элемент
-2
Введите [ 1 , 1 ] элемент
-1
Введите [ 1 , 2 ] элемент
0
Введите [ 1 , 3 ] элемент
4
Введите [ 2 , 0 ] элемент
-5
Введите [ 2 , 1 ] элемент
4
Введите [ 2 , 2 ] элемент
5
Введите [ 2 , 3 ] элемент
-5
Исходный массив:
-1 5 4 -5
-2 -1 0 4
-5 4 5 -5
Полученный массив:
0 1 1 0
0 0 0 1
0 1 1 0

```

Вариант 1.

1. Вычислить сумму и число положительных элементов матрицы $A[N, N]$, находящихся над главной диагональю.
2. Дана матрица $B[N, M]$. Найти в каждой строке матрицы максимальный и минимальный элементы и поменять их с первым и последним элементами строки соответственно.

Вариант 2.

1. Дана целая квадратная матрица n -го порядка. Определить, является ли она магическим квадратом, т. е. такой матрицей, в которой суммы элементов во всех строках и столбцах одинаковы.

2. Дана прямоугольная матрица $A[N, N]$. Переставить первый и последний столбцы местами и вывести на экран.

Вариант 3.

1. Определить, является ли заданная целая квадратная матрица n -го порядка симметричной (относительно главной диагонали).
2. Дана вещественная матрица размером $n \times m$. Переставляя ее строки и столбцы, добиться того, чтобы наибольший элемент (или один из них) оказался в верхнем левом углу.

Вариант 4.

1. Дана прямоугольная матрица. Найти строку с наибольшей и строку с наименьшей суммой элементов. Вывести на печать найденные строки и суммы их элементов.
2. Дана квадратная матрица $A[N, N]$, Записать на место отрицательных элементов матрицы нули, а на место положительных — единицы.
Вывести на печать нижнюю треугольную матрицу в общепринятом виде.

Вариант 5.

1. Упорядочить по возрастанию элементы каждой строки матрицы размером $n \times m$.
2. Дана действительная матрица размером $n \times m$, все элементы которой различны. В каждой строке выбирается элемент с наименьшим значением. Если число четное, то заменяется нулем, нечетное — единицей. Вывести на экран новую матрицу.

Вариант 6.

1. Дана целочисленная квадратная матрица. Найти в каждой строке наибольший элемент и в каждом столбце наименьший. Вывести на экран.
2. Дана действительная квадратная матрица порядка N (N — нечетное), все элементы которой различны. Найти наибольший элемент среди стоящих на главной и побочной диагоналях и поменять его местами с элементом, стоящим на пересечении этих диагоналей.

Вариант 7.

1. Квадратная матрица, симметричная относительно главной диагонали, задана верхним треугольником в виде одномерного массива. Восстановить исходную матрицу и напечатать по строкам.
2. Для заданной квадратной матрицы сформировать одномерный массив из ее диагональных элементов. Найти след матрицы, просуммировав элементы одномерного массива. Преобразовать исходную матрицу по правилу: четные строки разделить на полученное значение, нечетные оставить без изменения.

Вариант 8.

1. Задана матрица порядка n и число k . Разделить элементы k -й строки на диагональный элемент, расположенный в этой строке.
2. Задана квадратная матрица. Получить транспонированную матрицу (перевернутую относительно главной диагонали) и вывести на экран.

Вариант 9.

1. Для целочисленной квадратной матрицы найти число элементов, кратных k , и наибольший из этих элементов.
2. В данной действительной квадратной матрице порядка n найти наибольший по модулю элемент. Получить квадратную матрицу порядка $n - 1$ путем отбрасывания из исходной матрицы строки и столбца, на пересечении которых расположен элемент с найденным значением.

Вариант 10.

1. Найти максимальный среди всех элементов тех строк заданной матрицы, которые упорядочены (либо по возрастанию, либо по убыванию).
2. Расположить столбцы матрицы $D[M, N]$ в порядке возрастания элементов k -й строки ($1 \leq k \leq M$).

Вариант 11.

1. В данной действительной квадратной матрице порядка n найти сумму элементов строки, в которой расположен элемент с наименьшим значением. Предполагается, что такой элемент единственный.
2. Среди столбцов заданной целочисленной матрицы, содержащих только такие элементы, которые по модулю не больше 10, найти столбец с минимальным произведением элементов и поменять местами с соседним.

Вариант 12.

1. Для заданной квадратной матрицы найти такие k , что k -я строка матрицы совпадает с k -м столбцом.
2. Дана действительная матрица размером $n \times m$. Требуется преобразовать матрицу: поэлементно вычесть последнюю строку из всех строк, кроме последней.

Вариант 13.

1. Определить наименьший элемент каждой четной строки матрицы $A[M, N]$.
2. Найти наибольший и наименьший элементы прямоугольной матрицы и поменять их местами.

Вариант 14.

1. Задана квадратная матрица. Переставить строку с максимальным элементом на главной диагонали со строкой с заданным номером m .
2. Составить программу, которая заполняет квадратную матрицу порядка n натуральными числами $1, 2, 3, \dots, n^2$, записывая их в нее «по спирали».

Например, для $n = 5$ получаем следующую матрицу:

1 2 3 4 5

16 17 18 19 6

15 24 25 20 7

14 23 22 21 8

14 12 11 10 9

Вариант 15.

1. Определить номера строк матрицы $R[M, N]$, хотя бы один элемент которых равен c , и элементы этих строк умножить на d .
2. Среди тех строк целочисленной матрицы, которые содержат только нечетные элементы, найти строку с максимальной суммой модулей элементов.