СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Интернет как система начинает выступать для его пользователя как «коллективный разум». Из Интернета пользователь может получить сегодня для своего ума не только информационные данные, но и интеллектуальную помощь в их осмыслении, эмоциональное сопереживание в азартной «on-line» игре. Сеансы «общения» с Интернет формируют жизненные ориентиры. Поэтому сегодня уже появился термин «психологические феномены Интернет-технологий», а в университетах читается соответствующий курс лекций. Психологи и Web-программисты взаимодействуют. В их обиходе используются понятия «Интернет –аддикция», феномен потока, эффект присутствия, азарт, риск, аутизм.

Данное методическое пособие рассматривает психологические вопросы Интернет-технологий и вопросы Web-программирования, но которые даются студенту в предположении, что он изучил психологический аспект. В этом состоит особенность специальности «психологическое обеспечение информационных технологий», по которой идет подготовка студентов на кафедре ИПиЭ.

Точно так же, как психолог обязан знать строение и функции мозга, принципы работы нейрона, точно так же сегодня он обязан знать Интернет-программирование, т.е. язык и «нейроны» как материальный субстрат «коллективного разума». В психологической науке есть отрасль «дифференциальная психология», которая изучает зависимость психики человека от анатомических особенностей его мозга. Аналогично и языковые средства, структура таблиц, кодов, баз данных, с помощью которых работает конкретный сайт, сервер, влияют на эргономику работы человека с этим Интернет-приложением, на усталость, обучаемость, ход мышления пользователя. Вышесказанное поясняет, почему будущему специалисту в области компьютерного проектирования необходимо изучить данное пособие по курсу «психологические феномены Интернет-технологий».

1 Создание простейшего HTML-документа (2 ч)

Цель: формирование практических умений по созданию простейших HTML-документов.

1.1 *Теоретические сведения*

Для форматирования Web-документов обычно используют язык гипертекстовой разметки − HTML (HyperText Markup Language). Web-документ представляет из себя обычный текст (в противоположность двоичным данным), который может быть прочитан человеком. Созданный таким образом,. Web-документ является HTML-документом и имеет расширение .html. Для его просмотра нужно пользоваться браузером (Internet Explorer, Mozilla Firefox, Opera).

Разметка частей HTML-документа представляет собой семантическое выделение структурных блоков текста: заголовков, параграфов и т.д. Разметка осуществляется с помощью специальных маркеров, или тегов(например <b>), которые заключаются в угловые скобки (<>). Тег, имеющий некоторое содержимое (возможно, и другие теги), − называется *контейнером*. В этом случае принято выделять открывающий (начальный) тег и закрывающий (конечный) тег, которые имеют следующий вид:

<тег>... содержимое ... </тег>, т. е. закрывающий тег идентичен открывающему за исключением символа косой черты перед именем тега.

В языке HTML регистр символов не учитывается, но в языке-наследнике XHTML как подмножестве языка XML регистр символов имеет значение. Поэтому для соблюдения обоих языковых стандартов при создании документов рекомендуется использовать строчный регистр.

Итак, простейший HTML-документ будет иметь следующий вид:

```
<html>
  <head>
    <title>Hello, world!</title>
  </head>
  <body>
    <p>Hello, world!</p>
  </body>
</html>
```

Тэг <html> является корневым контейнером для для всего документа. Он должен включать в себя заголовок (<head>) с названием документа (<title>), а также содержимое (тело) документа (<body>). В тегах тела документа представлено описание содержимого документа. Тег <body> является тегом блочного уровня, который может содержать только блочные элементы. В работе ограничимся следующими блочными элементами: заголовками h1 (h2, h3 ... h6) и p (параграфом). Элемент h1 представляет собой заголовок первого уровня (самый логически важный), который может содержать только текст (и не может другие

заголовки или параграфы). С аналогичной целью применяют заголовки следующих уровней (различие между ними лишь семантическое −заголовок второго уровня считается менее важным, чем заголовок первого уровня и т.д.).
Тег «Параграф» ( ) может содержать текст, но не может содержать заголовки или другие параграфы.

Для форматирования отдельных частей текста существуют теги *строчного* уровня:

 − bold (делает начертание текста **жирным**);

 − italic (делает начертание текста *наклонным*);

<u> − underline (подчёркивает текст);

 − emphasize (логически выделяет некоторую часть текста);

 − ставит сильное логическое ударение по сравнению с ;

Использование тегов для визуального форматирования (, , <u>) не приветствуется, т. к. на современном этапе развития Web-программирования для визуального представления информации применяется другая технология − CSS. Лучше использовать теги для логического форматирования.

Строчные контейнеры можно вкладывать друг в друга, например:

  Привет!

В HTML-документе такая разметка отобразит абзац текста, где слово «Привет» будет одновременно начертано жирным и наклонным шрифтом.

По стандарту HTML все пробельные символы (whitespaces) − собственно сам пробел, символ табуляции, символ переноса строки и возврата каретки, сжимаются до величины одного пробела. Если такое форматирование нежелательно, то используют тег <pre> (preformatted text − предформатированный текст). Он является контейнером блочного уровня.

Для разделения частей документа горизонтальной линией существует тег без содержимого <hr /> (horizontal rule − горизонтальная линейка). Он не может быть вложенным в параграф.

1.2 *Порядок выполнения работы*

Создать HTML-документ, в котором будет представлена ваша автобиография в произвольном стиле изложения. Обязательные требования к документу: автобиография должна включать в себя несколько абзацев, каждый из них должен иметь заголовок − например «Детство»; абзацы должны быть разделены горизонтальной линейкой. Ключевые моменты вашего жизненного пути также должны быть выделены.

1.3 *Контрольные вопросы*

1 Какие браузеры вы знаете?

2 В чём отличие тегов визуального форматирования от тегов семантического форматирования?

3 Для чего нужен тег <pre>?

4 В чём отличие между тегом <pre> и тегом ?

2 Работа с изображениями и ссылками в HTML (2 ч)

Цель: формирование практических навыков встраивания изображений и гиперссылок в HTML-документы.

2.1 *Теоретические сведения*

Восприятие («перцепция», от лат. perceptio − представление, восприятие) - сложный процесс приема и преобразования сенсорной информации, формирующий субъективный целостный образ объекта, воздействующего на анализаторы через совокупность ощущений, инициируемых данным объектом.

На сегодняшний день имеются следующие основные свойства зрительного восприятия: последовательность, избирательность, целостность, запоминаемость, константность, соотносительность, иллюзорность, ассоциативность, образность. Существенный вклад в понимание того, как из отдельных зрительно воспринимаемых деталей предметов складывается их целостная картина - образ, внесли представители гешталыппсихологии - направления научных исследований, сложившегося в начале XX в. в Германии.

Геометрические иллюзии (иллюзии Мюллера-Лайера) – наиболее часто изучаемые иллюзии зрительного восприятия. Большинство известных геометрических иллюзий можно рассматривать либо как искажение в восприятии величины (длины или размера), либо как искажение в восприятии направления линий. Лучшим примером иллюзии длины отрезка является иллюзия Мюллера-Лайера: две линии равной длины, одна из которых оканчивается сходящимися, а другая - расходящимися клиньями, воспринимаются человеком как неравные по длине. При этом эффект иллюзии настолько устойчив, что она возникает и в том случае, если человек знает о причинах ее возникновения.

Разделим зрительную информацию на текстовую и графическую.

Представление текстовой информации рассмотрено на предыдущем практическом занятии, поэтому далее рассмотрим визуальное представление графической информации в глобальной сети Интернет.

Графика бывает двух видов: векторная и растровая. Для векторной графики характерно задание изображений с помощью математических формул. Ее преимущество в том, что при масштабировании изображения (увеличении, уменьшении) не происходит потеря информации. Для растровой графики характерно «поточечное» хранение информации, так что потеря информации очень заметна при увеличении. В настоящее время в Интернете широкой поддержкой пользуется только растровая графика.

Изображения растровой графики могут иметь следующие форматы:

GIF − считается устаревшим на сегодняшний день форматом хранения изображений: поддерживает до 256 цветов, при этом один из цветов может быть прозрачным. В формате используется сжатие без потери информации. В

формате GIF могут храниться скриншоты и простейшие графики. Поддерживает анимацию.

PNG – служит более современным аналогом формата GIF: поддерживает миллионы цветов и произвольную прозрачность (alpha layer — от 0 до 100%).

JPEG – формат, который хранит информацию с потерей данных. Используется в основном для хранения фотографий, т.к. нужно выбирать между качеством изображения и его размером в байтах, находящимися в отношениях обратно пропорциональной зависимости.

Для встраивания изображений в HTML-документы предназначен тег <img />, который не имеет содержания. Тег по умолчанию <img /> является тегом строчного уровня. Для того, чтобы встроить изображение в документ, нужно использовать тег следующим образом:

Атрибуты src и alt являются обязательными.

Например, для загрузки изображения my.jpg, которое находится в той же директории, что и документ my.html, нужно в последнем использовать следующую конструкцию:

 ...

Для того, чтобы вокруг изображения располагался текст, нужно использовать CSS-свойство float (использование устаревшего атрибута align нежелательно), которое может принимать значения left (изображение находится слева, а текст обрамляет изображение справа) и right (изображение находится справа, а текст обрамляет изображение слева):

 ...

Внимание! Изображение обрамляет только тот текст, который находится внутри того же параграфа.

Однако в настоящее время наиболее популярно совместное использование информационных изображений и текста – инфографика.

Для размещения ссылок предназначен тег <a>. Он является элементом строчного уровня и может содержать в себе текст или изображение, по нажатию на которые осуществляется переход по ссылке. При этом ссылка может быть как на другой HTML-документ, так и на любой другой ресурс, например изображение. Для задания ссылки используется атрибут href, в котором указывается адрес ссылки (относительный либо абсолютный). Для абсолютных адресов характерно наличие протокола и имени хоста (либо ip-адреса) в URL; в относительных адресах эти параметры наследуются от текущего документа, в котором использована ссылка.

Например:

Поиск – ссылка на поисковый движок Google, которая будет показана в браузере просто как слово «Поиск». В данном случае используется абсолютная ссылка.

 – картинка index.gif используется в качестве относительной ссылки на документ index.html.

Существует возможность задания ссылок на части документа. Для этого нужно:

1 Пометить то место, на которое осуществляется ссылка.

2 Задать ссылку на помеченную область.

Первая из указанных процедур также осуществляется с помощью тега <a>. Для этого случая, используется атрибут id, который должен быть уникальным в пределах документа для идентификации области. Например, для маркирования параграфа, описывающего итальянские пиццы, нужно сделать так:

Итальянские пиццы изначально были пищей беднейших слоёв населения...

Идентификатор области, задаваемый атрибутом id, может быть произвольным, главное, чтобы он содержал только буквы латинского алфавита и цифры без пробелов.

Для задания ссылки на данный параграф можно сделать так:

итальянская пицца

Здесь в качестве значения атрибута href использован символ # и идентификатор. Ссылки на другие документы и ссылки внутри документа можно комбинировать, т. е. можно сослаться на определенную часть какого-либо документа. Для этого нужно после имени документа использовать символ # и идентификатор раздела. Например:

Итальянская пицца

2.2 *Порядок выполнения работы*

1 Создать 3 html-документа с изображениями (фотографии друзей, семьи, любимых героев) и кратким описанием к ним.

2 Создать файл index.html, включающий ссылки на эти три файла и на файл с вашей автобиографией (см. практ. занятие №1).

3 Модифицировать все файлы с изображениями таким образом, чтобы по нажатию на изображение осуществлялся бы переход к документу index.html.

4 Дополнить автобиографию содержанием, включающим названия соответствующих разделов: «Детство», «Отрочество», «Увлечения» и т. п. Каждая строка содержания должна ссылаться на соответствующей раздел вашей автобиографии. В конце каждого раздела должна стоять ссылка «К содержанию».

2.3 *Контрольные вопросы*

1 Какие форматы графики вы знаете?

2 В чём отличие растровой графики от векторной?

3 В чём отличие относительных ссылок от абсолютных?

4 Как можно по нажатию на изображение осуществить переход к некоторому документу?

5 Обоснуйте выбор фотографий и описание к ним с точки зрения:

a) веб-программиста;
b) пользователя.

3 Работа с таблицами и списками в HTML (2 ч)

Цель: формирование практических навыков использования списков и таблиц.

3.1 *Теоретические сведения*

При восприятии человеком сложных осмысленных изображений срабатывает механизм влияния прошлого опыта и мышления, выделяющий в воспринимаемом изображении наиболее информативные места, на основе которых, соотнеся полученную информацию с памятью, можно о нем составить целостное представление. Анализ записей движений глаз, проведенный А.Л. Ярбусом, показал, что элементы плоскостных изображений, привлекающих внимание человека, содержат участки, несущие в себе наиболее интересную и полезную для воспринимающего информацию.

Сеть переполнена списками, в том числе элементами, которые пользователь обычно не относит к спискам, например навигацией. Навигационная панель (Navigation Bar − Navbar) является списком ссылок. Пользователю легче создать общее представление о тексте и быстрее найти информацию, если она представлена в форме списка. Список − это набор пунктов, которые обладают одинаковым назначением и(или) схожими характеристиками. Наиболее эффективно использование простых списков

Все вышесказанное способствует созданию более удобных списков и таблиц для пользователя.

В HTML-стандарте предусмотрены следующие виды списков: список определений, упорядоченный список и неупорядоченный список.

Список определений состоит из одного и более определений. Определение состоит из определяемого слова (термина) и его описания. Пример списка определений:

```
<dl>
        <dt>Интернет</dt>
        <dd>
                <p>Глобальная сеть, объединяющая компьютеры по всему Земному шару...</p>
        </dd>
        <dt>Веб-сервер</dt>
        <dd>
                <p>Программное обеспечение, принимающее на вход HTTP-запросы и ...</p>
        </dd>
<dl>
```

Здесь список определений задаётся тегом <dl> (Definition List); определяемое слово задаётся тегом <dt> (Definiton Term); а развёрнутое описание задаётся с помощью тега <dd> (Definition Description). Все перечисленные элементы по умолчанию являются элементами блочного уровня, т. е. будут занимать одну «строку» (параграф) каждый:

Интернет

Глобальная сеть, объединяющая компьютеры по всему Земному шару...

Веб-сервер

Программное обеспечение, принимающее на вход HTTP-запросы и ...

Для упорядоченных списков семантически важен порядок следования элементов. Например, можно провести аналогию с приготовлением пирога: сначала нужно сделать тесто, добавив ингредиенты в определенной последовательности, а затем его запекать. Для неупорядоченных списков порядок следования элементов не имеет значения. В списке ингредиентов для пирога порядок их следования не важен.

Для задания упорядоченного списка объектов существует тег <ol> (Ordered List), а для неупорядоченного – (Unordered List). Каждый объект списка должен быть помещён в контейнер объекта списка (неважно какого, упорядоченного или неупорядоченного) – (List Item). Таким образом, для задания упорядоченного (неупорядоченного) списка нужно сначала задать контейнер объектов, а затем каждый объект «упаковать» в контейнер объекта. Например, список ингредиентов можно задать так:

```
<ul>
    <li>200 грамм муки</li>
    <li>50 грамм дрожжей</li>
    <li>Стакан воды</li>
</ul>
```

При просмотре документа в браузере упорядоченный список определений по умолчанию нумеруется с помощью арабских цифр (допустимы также строчные и прописные буквы греческого и римского алфавитов); для неупорядоченного списка определений используются точечки, квадратики, кружочки и т. п.

Списки могут быть «вложены» один в другой, т. е. некоторый список может являться объектом (или частью объекта) другого списка. В этом случае вложенный список должен быть заключён в тег <li>. Например, для задания следующего списка:

1. Продукты
    - продукты питания
    - продукты промышленности
2. Услуги
    - пошив одежды
    - монтаж шин

нужно написать следующий html-код:

```
<ol>
    <li>Продукты
        <ul>
            <li>продукты питания</li>
            <li>продукты промышленности</li>
        </ul>
    </li>
    <li>Услуги
        <ul>
            <li>пошив одежды</li>
            <li>монтаж шин</li>
        </ul>
    </li>
</ol>
```

Таблицы − структурный элемент HTML-документа, разработанный для представления информации в табличной форме. Разметка таблицы включает несколько компонентов и атрибутов, которые могут быть использованы для идентификации элементов в стандартной таблице данных: заголовки столбцов, строк, надписи и краткое изложение содержания. В связи с существующими ограничениями, накладываемыми средой - низкое разрешение, ограниченные возможности отображения и его многовариантность, - простые таблицы получили более широкое распространение в Сети.

Представить данные в виде строк и столбцов можно при помощи таблицы (тег <table>). Таблица состоит из строк (<tr> − table row), а строка состоит из ячеек (<td> − table data) или заголовков (<th> − table header). Различие между ячейками и заголовками должно быть чисто семантическим: заголовок задаёт название для всего столбца или строки, например, название продукта; а конкретные продукты должны тогда быть уже ячейками. Для того чтобы были видны границы таблицы можно использовать атрибут border со значением 1. Пример простейшей таблицы:

```
<table border='1'>
    <tr>
        <th>Название месяца</th>
        <th>Количество дней</th>
    </tr>
    <tr>
        <td>Декабрь</td>
        <td>31</td>
    </tr>
    <tr>
        <td>Ноябрь</td>
        <td>30</td>
```

```
        </tr>
    </table>
```

В качестве содержимого ячейки может быть использован отдельно текст, параграф, изображение и даже таблица (допускается вложенность таблиц).

3.2 *Порядок выполнения работы*

1 Оформить свою автобиографию в сокращённом варианте (как в анкетах) используя списки (например, место рождения: Минск). Использовать вложенные списки (например, для оформления увлечений).

2 Используя таблицы сформировать HTML-документ со своим расписанием или распорядком дня.

3 Используя вложенные таблицы, сформировать HTML-документ, где будут перемножены две невырожденные неединичные матрицы размерностью 4x4.

3.3 *Контрольные вопросы*

1 Какие бывают типы списков?

2 Приведите пример семантического различия упорядоченного списка от неупорядоченного.

3 Могут ли списки быть вложенными?

4 Может ли быть изображение объектом списка?

5 Как задаются таблицы?

6 Могут ли таблицы быть вложенными?

7 Оцените удобство восприятия созданного Вами HTML-документа. Определите его достоинства и недостатки.

4 Работа с формами (2 ч)

Цель: формирование практических умений создания форм в HTML-документах.

4.1 *Теоретические сведения*

Для того, чтобы сделать сайт интерактивным предназначены формы. Примеры использования форм: строка поиска в поисковой системе или форма регистрации пользователя. Формы - лишь средство для сбора информации, а удовольствие доставляют простота использования и правильность заполнения. Они должны располагаться в логичном и предсказуемом порядке, начиная с личных данных - имя, фамилия, дата рождения, пол и т.д., за которыми следует контактная информация - адрес, номер телефона, электронный адрес. Пользователи не должны повторно вводить одну и ту же информацию. Поля форм необходимо снабдить ясной маркировкой, а те поля, в которые нужно вводить сведения в определенном формате, следует сопроводить понятными указаниями. Формы могут состоять из множества различных элементов: меток, полей ввода,

флажков с независимой фиксацией (check box), кнопок переключателей (radio button) и выпадающих меню. Именно связь между элементами обеспечивает осмысленность и логичность любой формы и помогает пользователю заполнить ее. Для достижения визуальной логики связанные элементы обычно объединяются в группы.

Документ может содержать несколько форм. Элементом-контейнером форм является тег блочного уровня <form>, который может содержать в себе только элементы блочного уровней. Его атрибутами являются:

- method — определяет тип пересылки данных формы. Может принимать значения post или get. При отправке файлов обязательно должен быть установлен в значение 'post'.
- action — определяет серверный обработчик формы. В данной лабораторной работе можно использовать символ # в качестве значения
- name — задаёт имя формы (нужно для идентификации формы на стороне сервера, т.к. возможно наличие нескольких форм в документе).

Пример простейшей формы:

```
<form action='process.php' method='post'>
...
</form>
```

Те элементы, посредством которых пользователь может вводить информацию, называются управляющими элементами. Особенностью управляющих элементов является их возможность получать фокус. Управляющие элементы могут содержаться только внутри формы. Содержимое управляющих элементов — это те значения, которые ввёл пользователь.

Большинство управляющих элементов задаётся посредством элемента <input />. Данный элемент не может иметь вложенные элементы, т.е. он не является контейнером. Основные атрибуты:

- maxlength − задаёт максимальную длину строки для текстовых полей.
- name − обязательное поле, которое будет использоваться сервером для обработки отосланных содержимых управляющих элементов.
- type − определяет тип управляющего элемента. Может принимать следующие значения:

text − задаёт однострочное текстовое поле для ввода небольшого количества информации (имени, адреса и т.п.)

password − задаёт текстовое поле для ввода пароля. Обычно, вводимый символ отображается символом * для т.н. «визуальной безопасности».

checkbox − задаёт квадратную область, которая может иметь состояние «включено» или «выключено».

radio − задаёт переключатель, который иногда называется «радиокнопка».

Название взято по аналогии со старыми автомобильными приёмниками, в которых при нажатии на одну кнопку другая автоматически выключалась. Для задания группы переключателей (в которой только одно значение может быть выбрано) нужно задать требуемое количество элементов <input>, у которых

одно и тоже имя (имя группы, например «страна») − атрибут name, но разные значения − атрибут value.

submit − создаёт кнопку отправки данных формы на сервер. Надпись на форме задаётся значением атрибута value.

- value − задаёт изначальное значение управляющего элемента
- checked = 'cheched' − данный атрибут с данным значением сообщает браузеру, что состояние некоторого управляющего элемента «включено» (действительно для кнопок-переключателей и радиокнопок)

Для создания выпадающего меню-выбора можно использовать элемент <select>. Как и для элемента <input>, присутствие атрибута name обязательно. Для того, чтобы позволить пользователю выбрать сразу несколько пунктов меню, нужно добавить атрибут multiple='multiple'. Элемент-контейнер <select> может содержать только опции − тег <option>. Например:

```
<select name='size'>
    <option>Small</option>
    <option>Medium</option>
    <option>Large</option>
</select>
```

Для создания многострочной текстовой области ввода текста существует контейнер <textarea> (элемент блочного уровня), который может содержать только текст (начальное значение, которое показывается пользователю). Обязательными атрибутами данного контейнера являются:

- name − см. описание элемента в теге <input>;
- rows − задаёт количество строк в управляющем элементе;
- cols − задаёт количество столбцов в управляющем элементе.

Для логической группировки управляющих элементов предназначен тег <fieldset>, внутри которого можно задать тег <legend> для названия некоторой логической группы. Например,

```
<form action='process.php' method='post'>
    <fieldset>
        <legend>Подписка</legend>
        <label for='email'>Email:</label>
        <input type='text' id='email' name='email' />
        <input type='submit' name='subscribe' value='Подписаться' />
    </fieldset>
</form>
```

Для задания надписей для управляющих элементов используется тег <label> с обязательным атрибутом for, указывающим на идентификатор управляющего элемента (поле id). Особенно удобно для кнопок-переключателей (checkbox) и радиокнопок.

4.2 *Порядок выполнения работы*

Создать HTML-документ с одной формой, состоящей из трёх блоков: личная информация, предпочтения и обратная связь. В личной информации должны быть поля для ввода имени, фамилии, отчества, даты рождения, желания подписаться на список рассылки и пол. Во втором блоке должно быть два меню-списка с предпочтениями (продуктов, фильмов и т.п.), где во втором списке должна быть возможность для выбора нескольких пунктов. В блоке для обратной связи должна быть возможность для ввода многострочного комментария. В конце формы должна быть кнопка для отправки формы на сервер.

4.3 *Контрольные вопросы*

1 Какие существуют элементы управления для ввода текстовой информации?
2 В чём преимущество использования элемента <label>?
3 В чём отличие элемента от элемента <fieldset>?
4 Может ли элемент управления находиться вне контейнера <form>?
5 Какие психологические аспекты реализованы в Вашем HTML-документе?

5 Работа с таблицами стилей (4 ч)

Цель: формирование практических умений применения каскадных таблиц стилей.

5.1 *Теоретические сведения*

Каскадные таблицы стилей (CSS - cascade style sheets) предназначены для презентационного оформления HTML-документа. Средства CSS позволяют управлять отображением страницы и в то же время сохранять границу между содержанием и его представлением. Отпадает необходимость в использовании разметки для представления содержания и тех искусственных приемов, на которые полагались раньше, чтобы получить возможность управлять визуальным отображением Web-сайта. Теперь можно разрабатывать Web−страницы с использованием структурных элементов HTML, а для разработки их визуального отображения применять CSS. Можно разделить содержание и его представление, работая с содержанием в структурированных документах HTML, а с представлением - в таблицах стилей. Когда работа над содержанием и его представлением ведется раздельно, можно избежать противоречий между дизайнерскими решениями и реализацией доступа. Одним пользователям требуется доступ только к содержанию, другим необходима возможность настройки отображения содержания при осуществлении доступа. Если на страницах нет разметки для представления содержания, доступ к содержанию – со стилями дизайнера; стилями, выбранными пользователем; стилями программного обеспечения; или без стилей – может быть обеспечен в соответствии с требованиями пользователя. Существуют отличные от визуального представления, но, т.к. последнее является наиболее распространённым и поддерживаемым браузерами, то в дан-

ном методическом пособии рассматриваются только визуальные аспекты представления.

Таблицы стилей могут применяться следующими способами:

1) подключаться посредством встроенных в элемент стилей: нужно использовать атрибут style какого-либо элемента, причём в нём нужно писать только декларации, которые применяются к текущему элементу. Например, для того, чтобы сделать абзац зелёным цветом, нужно:

```
<p style='color: green'>Hello, world!</p>
```

2) подключать посредством встроенных в документ стилей: используется тег <style> внутри секции head:

```
<html>
    <head>
        <title>...</title>
        <style type='text/css'>
        /* Стили...
        */
        </style>
    </head>
...
```

3) подключать посредством внешних файлов CSS: используется тег <link> в секции head: <link type='text/css' rel='stylesheet' href='main.css' />

Каскадные таблицы стилей состоят из множества правил CSS. Правило CSS состоит из селектора и заключённых в фигурные скобки {} деклараций, разделённых точкой с запятой:

```
p { color: blue;}
```

Здесь p — селектор; color: blue; - декларация.

Декларация содержит собственно презентационные аспекты по схеме «свойство: значение».

Существуют следующие типы селекторов:

- универсальный селектор * − выбирает все элементы;
- селектор элементов − выбирает все элементы по имени тега: p {color: blue;} - делает все абзацы синего цвета;
- селектор класса − выбирает все элементы, с соответствующим значением атрибута класс: .info {color: red; } − делает все элементы, имеющие класс info красного цвета;
- селектор идентификатора − выбирает элемент, который имеет некоторый идентификатор: #hi {color: black;}.

Если несколько правил совпадают, то применяется наиболее специфическое. Наиболее специфическим считается правило, которое охватывает наименьшее количество элементов. Очевидно, что наиболее специфическим является селектор идентификатора, а наименее специфическим является универсальный селектор.

Селекторы могут комбинироваться, что увеличивает специфичность. Например,

p.info − осуществляет выбор всех параграфов с классом info; возможна выборка элементов с помощью задания селектора-родителя;

p.info − em {color: silver} выбирает все элементы em, которые находятся внутри параграфов с классом info; количество элементов-предшественников не ограничено.

Наиболее употребляемые единицы измерения в CSS:

px (пиксель, точка на экране),

% (относительная величина, задающая размер относительно соответствующих размеров элемента-контейнера, т. е. width: 20 % задаёт ширину элемента, равную 20 % от элемента контейнера),

em (относительная величина, задающая размер относительно текущего шрифта). Указание единиц измерения является обязательным.

Способы задания цветов:

1 Использовать англоязычное название цвета (color: black).

2 Использовать шестнадцатеричную нотацию задания цвета #RRGGBB (color: FFFFFF).

3 Используя десятеричную нотацию задания цвета rgb(r,g,b) (color: rgb(255,255,255)).

Перечислим некоторые визуальные аспекты презентации:

- color − задаёт основной цвет текста;
- background-color − задаёт цвет фона;
- background-image − задаёт фоновый рисунок (backgound-image: url (pizza.jpg));
- border-width − задаёт толщину границ;
- border-style − задаёт стиль обрисовки границ (none, dashed — штриховой линией, dotted − пунктирной, solid — сплошной);
- border-color − задаёт цвет границ;
- padding − задаёт внутренний отступ (padding: 5px);
- margin − задаёт внешний отступ для элемента;
- text-align − задаёт расположение текста (center, left, right, justify);
- text-indent − задаёт отступ красной строки для параграфа (text-indent: 2.5 em);
- font-size − задаёт размер шрифта;
- font-weight − задаёт жирность начертания шрифта (от 100 до 900);
- font-style − задаёт наклонность шрифта (normal, italic);
- font-family − задаёт семейство шрифтов (Times, Arial, Verdana).

5.2 *Порядок выполнения работы*

1 Используя все три способа подключения CSS и все вышеперечисленные свойства, представить свою автобиографию в удобном для восприятия виде.

2 Создать списки (упорядоченные или неупорядоченные) с трёхкратным уровнем вложенности друг в друга (например, продукты → масла → сливочные, оливковые) и задать различные цвета фона для каждого уровня вложенности в документе HTML.

5.3 *Контрольные вопросы*

1 Для чего предназначена технология CSS?

2 В чём преимущество CSS по сравнению с презентационными тегами и атрибутами HTML?

3 Какие существует единицы измерения в CSS? Какие из них являются относительными, а какие − абсолютными?

4 Для чего предназначен селектор?

5 Что обозначает уровень специфичности селектора?

6 Перечислите основные типы селекторов.

7 Проведите психологический анализ разработанного вами HTML-документ по следующим пунктам:

a) цветовое решение документа;
b) выбор шрифта и его характеристик (размер, начертание и т. д.);
c) оформление графических элементов документа;
d) расположение компонентов документа;

Является ли разработанный вами HTML-документ удобным для пользователя (клиента)?

6 Создание простейшей php-программы (2 ч)

Цель: формирование практических умений создания простейших программ на языке PHP, понимание принципов работы PHP-интерпретатора и его взаимодействия с веб-сервером.

6.1 *Теоретические сведения*

Язык PHP является императивным С-подобным языком. Язык PHP является интерпретируемым с проверкой типов во время исполнения. Поэтому код, написанный на языке PHP и сохранённый в виде файла с расширением php, называется *скриптом*.

Для отработки (выполнения) скрипта требуется его исполнение посредством PHP-интерпретатора. PHP-интерпретатор может быть вызван либо непосредственно с вашим участием, либо его вместо вас может вызывать web-сервер.

Допустим, что интерпретатор PHP проинсталлирован в директорию C:\php, а php-скрипт находится в директории d:\mylaba\ и называется test.php. Тогда для непосредственного вызова интерпретатора требуется выполнить следующие действия:

1 Запустить интерпретатор командной строки Windows (пуск → выполнить → cmd).

2 В интерпретатор командной строки вызвать интерпретатор PHP и передать ему в качестве аргумента полный путь к файлу-скрипту PHP (C:\php\php.exe d:\mylaba\test.php).

3 Если всё произошло успешно, то вы увидите вывод скрипта. Если в скрипте имеются ошибки, то интерпретатор сообщит вам о них.

Важно понять, что скрипт осуществляет вывод данных в так называемый стандартный поток вывода (stdin в стандартной библиотеке C). При запуске интерпретатора PHP из командной строки в качестве стандартного потока вывода будет сама консоль, а при запуске его из web-сервера стандартным потоком вывода будет некоторая область памяти, которая затем передаётся по сети пользователю и отображается в браузере, т. е. упрощённо говоря, вывод осуществляется «в браузер» пользователя. Таким образом, стандартный поток вывода задаётся внешним образом для интерпретатора и скрипта.

При написании скриптов в данной лабораторной работе следует учитывать стандартный поток вывода, т. е. осуществлять вывод в консоль без разметки HTML.

PHP-скрипт содержится в файле с расширением .php (строго говоря, расширение может быть любым и может быть перенастроено на любое другое; тем не менее, будем придерживаться общепринятых соглашений). Содержимое этого файла − текст. При отсутствии специальных вставок (инструкций для php-интерпретатора) данный текст напрямую дублируется в стандартный поток вывода. Специальные инструкции − это php-код, содержащийся в блоке, который начинается с символов **<?php** и заканчивается символами **?>**. Количество таких вставок в файле не ограничено. «Чистый» php-скрипт будет содержать только php-код, обрамлённый данными последовательностями в начале и в конце файла.

PHP-код состоит из инструкций, разделённых оператором «точка с запятой».

Одна из простейших инструкций − это echo, которая выводит свой аргумент-строку в стандартный поток вывода. Пример файла-скрипта:

```
<?php
 echo 'Hello, world!';
?>
```

Идентификатор переменной в PHP задаётся обычным способом (первый символ идентификатора — английская буква, остальное — комбинация из букв, цифр или символа). Сама переменная задаётся посредством предварения идентификатора символом доллара ($). Возможно, чтобы переменной тут же было присвоено значение. Например:

```
$a = 5;
$hw = 'PHP is cool';
```

Строки в PHP задаются двумя способами: либо в одинарных кавычках, либо в двойных. При задании строки в одинарных кавычках строка интерпретируется буквально (т. е. не осуществляется подстановка переменных и интер-

претация специальных символов). При задании строки в двойных кавычках указанные действия, наоборот, осуществляются, т. е.:

```
$a = 55;
$b = " a = $a\n ";
```

Таким образом, в переменной $b будет храниться значение a = 55 и символ начала новой строки.

Для перебора значений в ассоциативном списке существует операция foreach, которая пробегает весь массив, присваивая на каждой итерации соответствующее значение ключа и значения ключа из ассоциативного массива. Блок кода, который итерируется, заключается в фигурные скобки. Например:

```
foreach($arr as $key => $value)
{
//...
}
```

В данном случае осуществляется итерирование по массиву $arr, а в качестве ключа для каждой итерации будут использоваться переменные $key и $value для значений. Переменные $key и $value не нужно заранее объявлять.

Для доступа к данным формы, которые пользователь ввёл с формы, можно использовать массив $_REQUEST, который доступен в любом месте PHP-программы.

6.2 *Порядок выполнения работы*

1 При необходимости проинсталлировать комплекс программ WAMP или Denwer.

2 Разобраться, в какой директории следует размещать файлы, для того чтобы иметь к ним доступ через браузер.

3 Написать простейшую программу, выводящую текст «Hello, World!», и запустить её вручную с помощью интерпретатора PHP. Использовать переменные для хранения строк.

4 Написать простейшую программу, выводящую текст «Hello, World!» в виде HTML-документа, и запустить скрипт через web-server (т.е. просмотреть вывод программы в браузере).

5 Модифицировать HTML-страницу с формами, так, чтобы обработчик форм указывал на php-скрипт. Создать этот php-скрипт, который будет распечатывать все введённые в форму значения.

6.3 *Контрольные вопросы*

1 Что такое стандартный поток вывода?

2 В чём преимущество и недостатки интерпретируемых и слаботипизированных языков?

3 Что такое скрипт?

4 В чём отличие вывода результата отработки php-скрипта в консоль и «в браузер»?

7 Освоение управляющих конструкций PHP (2 ч)

Цель: формирование практических умений использования управляющих конструкций на примере решения математических задач.

7.1 *Теоретические сведения*

Управляющие конструкции − это конструкции, которые влияют на ход выполнения программы. Назовем основные из них.

exit − служит для немедленного выхода из программы, т. е. для завершения работы интерпретатора PHP. Последующее содержимое (находящееся после php-кода) не выводится в стандартный поток вывода.

die('message') − также служит для немедленного выхода из программы. В отличие от команды exit, эта инструкция принимает на вход строку-аргумент, которую выводит в стандартный поток вывода. Обычно эта команда используется для отладки.

Для отладки можно также использовать следующие функции:

print_r($var, [boolean out]) − производит рекурсивную распечатку переменной $var. Рекурсивность обозначает, что если в переменной содержатся другие структуры данных (массивы), то они также будут распечатаны. Если функция вызывается с одним аргументом, то вывод производится в стандартный поток вывода; если передан второй параметр, равный true, то функция возвратит строку, содержащую рекурсивную распечатку переменной $var.

var_dump($var, [Boolean out]) − функция, полностью аналогичная функции print_r, за исключением того, что здесь вывод осуществляется в формате html.

В php имеются стандартные для С операции, в том числе и операции комбинированные с присваиванием, т. е.:

```
$a = 5+2; // = 7
$b = 6-3; // =3
$c = $b*4; //=12
$d = $c/$b; // = 4
$e = $d % 3; // = 1 Операция взятия остатка от деления
$e += 20; // = 21
```

и т. д.

Имеются также унарные операции префиксного и постфиксного инкрементирования и декрементирования ($a++, —$a; ++$a, $a—).

Логические операции также С-подобные, например : && − логическое И, || − логическое ИЛИ. Они могут использоваться в конструкции if, полный синтаксис которой таков:

```
if(condition1){
        //block1
}elseif(condition2){
        //block2
...
```

```
}elseif(conditionN){
        //blockN
}else{
        //blockN+1
        }
```

Если выполняется условие condition1 (т. е. результат оценки выражения condition1 равен true), то выполняется инструкция в 1-м блоке (block1). Если же условие не выполняется, то проверяется следующее условие и т. д. Если ни одно из условий не выполняется, то будут выполнены инструкции в блоке else (blockN+1). Все части конструкции, кроме if, являются опциональными, т. е. их можно опустить:

```
if($a > 2){
        echo 'a is greater than 2';
}
```

Цикл while, как и в C, существует в двух вариантах: цикл while с предусловием и с постусловием.

```
while(condition1){
        //block1
}
do{
        //block2
}while(condition2);
```

В первом случае блок операций block1 выполняется до тех пор, пока condition1 истинно, причём condition1 проверяется до исполнения блока. То есть, теоретически возможен случай, что операции в блоке не выполнятся ни разу, если condition1 будет ложно.

Во втором случае проверка условия осуществится после исполнения блока инструкций block2. То есть, инструкции выполнятся как минимум один раз.

Цикл for в общем виде можно представить так:

```
for(<блок инициализации>; <условие>; <блок изменения счётчиков>)
{
//итерируемый блок.
}
```

Блок инициализации выполняется один раз до начала цикла. Обычно в нем присваиваются начальные значения переменных. Затем происходит проверка условий. Если условие возвращает результат true, то выполняются операции из итерируемого блока. По окончании выполняется блок изменения счётчиков и снова происходит проверка условия и переход на новую итерацию. В блоке инициализации и в блоке изменения счётчиков разделение операторов можно осуществлять только через запятую. Все блоки являются необязательными, т. е. при необходимости можно опустить часть из них. Пример простейшего цикла, распечатывающего значения от 10 до 20:

```
for($i = 10; $i < 20; $i++){
        echo 'i = ' + $i;
```

}

7.2 *Порядок выполнения работы*

1 Создать скрипт, который осуществит вывод первых 50 членов арифметической прогрессии $a_n = a_{n-1} + 15$, а также выведет сумму этих членов. Для ввода начального значения a_0 нужно создать отдельную страничку с формой. Вывод осуществлять в виде HTML.

2 Создать скрипт, который осуществит вывод первых 50 членов геометрической прогрессии $b_n = (-1)^n * 1.5$, а также суммы этих членов. Для ввода начального значения b_0 нужно создать отдельную страничку с формой. Вывод осуществлять в виде HTML.

3 Создать скрипт, выводящий корни квадратного уравнения $ax^2 + bx + c = 0$. Коэффициенты a, b, c должны вводиться на отдельной странице. При отсутствии решения вывести соответствующее сообщение.

7.3 *Контрольные вопросы*

1 Для чего предназначены управляющие конструкции?

2 Как еще следует осуществлять отладку php-скриптов?

3 В чём отличие конструкции if в PHP от аналогичной конструкции в языке C?

4 Как осуществляется явное приведение типов?

5 Как осуществляется неявное приведение типов?

8 Серверная обработка форм (2 ч)

Цель: формирование практических умений по обработке данных HTML-форм на стороне сервера с помощью языка PHP.

8.1 *Теоретические сведения*

Формы предназначены для интерактивного взаимодействия с пользователями. Нужно понимать принципиальное различие двух компонентов форм:

1 Форма, которая представляется пользователю, − HTML документ.

2 Когда пользователь отсылает данные формы на сервер − происходит их обработка на стороне сервера.

На стороне сервера могут выполняться следующие задачи:

1 Проверка правильности введённых данных в формы.

2 Осуществление бизнес-логики при корректности данных форм (внесение в базы данных и т. п.).

Формы могут отсылаться двумя способами − методом GET и методом POST. При отсылке «нечувствительной» информации можно использовать метод GET. При отсылке паролей желательно использовать метод POST. При отсылке файлов на сервер обязательно использовать метод POST.

Серверный обработчик форм задаётся атрибутом action элемента form, а метод задаётся атрибутом method:

<form action='myHandler.php' method='post'>

...

Из php-скрипта данные форм доступны через ассоциативные массивы $_GET и $_POST соответственно. Для унифицированного доступа к данным формы можно использовать переменную $_REQUEST. Ключами массивов являются названия управляющих элементов. Так, допустим в форме имеется:

<input type='text' name='first_name'>

Тогда то значение, которое ввёл пользователь, будет доступно как $_REQUEST['first_name'].

Допускается использование массивов (особенно удобно для управляющего элемента типа checkbox):

Предпочтение 1: <input type='checkbox' name='favorites[]' />

Предпочтение 2: <input type='checkbox' name='favorites[]' />

Тогда при обращении $_REQUEST['favorites'] − будет доступен массив, у которого первым элементом будет значение on, если пользователь выбрал первое предпочтение. Иначе говоря, проверить, выбрал ли пользовать первый элемент можно так:

if($_REQUEST['favorites'][0])

Для подключения внешних файлов к текущему PHP-скрипту можно использовать директивы require или include. Они отличаются тем, что в случае ошибки подключения внешнего файла require сгенерирурет фатальную ошибку и обработка сценария прекратится. Include же выведет только предупреждение об ошибке, но обработка скрипта будет продолжена. Например, подключить файл form.php можно так:

require('form.php');

8.2 *Порядок выполнения работы*

1 Модифицировать лабораторную работу по работе с HTML-формами (ввод личной информации − биографии): установить и создать свой серверный обработчик форм.

2 Проверить корректность введенных данных: если допущены ошибки, вверху формы вывести красным шрифтом список ошибок, снова вывести форму. Введенные пользователем и корректные значения должны быть установлены по умолчанию.

3 Если введенная информация корректна, то должен отобразиться текст − полноценная биография − связный текст, например, «Иванов Иван Иванович, родился 10.10.89 в городе Минске. Отдаёт предпочтение пирожкам и конфетам с маком и т.п.». Обратить особое внимание на окончания, которые зависят от выбранного пола.

8.3 *Контрольные вопросы*

1 Чем отличаются метод отсылки данных формы GET от метода POST?

2 Как идентифицируются введенные пользователем данные на стороне сервера?

3 В каких случаях удобно использовать в качестве идентификатора массивы?

4 Чем отличается require от include?

5 Как можно избежать вывода ошибок при подключении директивой include?

9 Обработка массивов в PHP (2 ч)

Цель: Приобретение практических навыков работы с массивами.

9.1 *Теоретические сведения*

В PHP массивы бывают двух типов − обычные и ассоциативные. В обычных массивах данные хранятся последовательно, а в качестве ключей для доступа к элементам используются индексы − порядковый номер элемента в массиве. Обычный массив задаётся с помощью ключевого слова array():

```
$myArray = array();
```

Можно сразу же задавать массивы «на месте»:

```
$myArray = array(1,2,3,5);
```

Доступ к элементам массива осуществляется посредством оператора [], принимающего на вход индекс. Индексы в PHP считаются с нуля, т. е. индекс первого элемента в списке − 0.

```
$myArray[0] = 'Hello';
```

Для получения количества элементов в списке (размера массива) используется функция length:

```
echo 'my array contains' + length($myArray) + ' elements';
```

Для удобства дополнения обычных массивов может использоваться оператор [] без параметров. Например, следующая конструкция дополняет массив $myArray значением 12:

```
$myArray[] = 12;
```

Обратите внимание, что массив не обязательно инициализировать или объявлять до использования, т. е. массив уже объявляется своим использованием:

```
$myNewArray[] = 2;
```

Если даже массива $myNewArray не существовало, то он будет создан, и значение 2 будет в него вложено.

Особенность ассоциативных списков в том, что в качестве индекса у них выступает строка. Такие списки обычно являются неупорядоченными. Создаются ассоциативные массивы точно так же, как и обычные массивы − с помощью оператора array(). Пример инициализации:

```
$student = array(
        'age'           => 20,
        'first_name'   => 'ivanov',);
```

В примере был создан ассоциативный массив $student, имеющий в качестве ключей строки age и first_name, а в качестве значений − число 20 и строку 'ivanov' соответственно. С точки зрения интерпретатора, отступы не обязательны но очень желательны для понимания кода другими пользователями. Особенно важны отступы при задании сложных структур.

Например, зададим массив (обычный) из имен студентов (ассоциативный массив):

```
$students = array(
        array(
                'age'           => 20,
                'first_name'   => 'ivan',
        ),
        array(
                'age'           => 21,
                'first_name'   => 'petia',
        ),);
```

Для итерирования по массивам может использоваться обычный цикл for. Есть упрощённая конструкция, которая итерирует по всем элементам массива без учёта индекса:

```
foreach($myarray as $value){
        echo 'value = ' + $value;}
```

В данном случае осуществляем итерацию по массиву $myarray, в качестве текущего элемента используется переменная $value.

Аналог для ассоциативных массивов:

```
foreach($myarray as $key => $value){
….}
```

9.2 *Порядок выполнения работы*

1 Создать ассоциативный массив, состоящий из двух обычных массивов.
2 Первый обычный массив заполнить геометрической прогрессией.
3 Второй обычный массив заполнить последовательностью чисел Фибоначчи.
4 Создать рекурсивные функции для распечатки массивов:
   1) вывод в консоль значений (одна строка, состоящая из индекса и значения);
   2) вывод в виде html-вывода.

9.3 *Контрольные вопросы*

1 Чем отличаются ассоциативные массивы от обычных?
2 Как осуществлять доступ к обычным массивам?

3 Как осуществлять доступ к ассоциативным массивам?
4 Как получить количество элементов в массиве?
5 Какое условие нужно соблюдать при написании рекурсивных функций?

10 Ознакомление с основными функциями PHP (2 ч)

Цель: приобретение практических навыков работы с функциями времени в PHP и декомпозиции обработчика на стороне сервера.

10.1 *Теоретические сведения*

Существует множество преимуществ от повторного использования программного кода. В PHP это достигается посредством выделения кода в отдельные функции. Функция − это именованный участок кода, который может быть вызван. Имя функции состоит из строчных и прописных букв английского алфавита, цифр и символа _. Функция определяется посредством ключевого слова function. Функция может возвращать значение посредством ключевого слова return.

Например, определим функцию сложения двух чисел:

```
function add($a, $b){
    $r = $a + $b;
    return $r;}
```

В PHP существуют две функции для подключения внешних функций: require и include. Обе функции имеют аналоги с постфиксом _once(), т. е. require_once и include_once. Обе функции подключают некоторый внешний файл и выводят его в стандартный поток вывода, интерпретируя PHP-код, находящийся в этом файле, если он обрамлён соответствующими маркерами (<?php ... ?>).

Функции require и include различаются по способу интерпретации ошибки подключения внешнего файла: require интерпретирует ошибку как фатальную, что приводит к завершению работы скрипта; inlcude интерпретирует ошибку как нефатальную, выводит предупреждающее сообщение, и скрипт продолжает выполнятся. Вывод ошибки можно подавить с помощью оператора @. В качестве аргумента функции принимают строку, содержащую имя подключаемого файла.

Например, для подключения внешнего файла funcs.php нужно использовать конструкцию:

```
require ('functs.php').
```

Постфикс _once сообщает интерпретатору, что если файл уже был однажды подключён, то его повторное подключение не осуществляется.

Для вывода даты используется функция date:

```
string date  ( string $format  [, int $ timestamp  ] )
```

Эта функция возвращает время, отформатированное в соответствии с аргументом format, используя метку времени, заданную аргументом timestamp или текущее системное время, если timestamp не задан. Другими словами,

timestamp является необязательным и по умолчанию равен значению, возвращаемому функцией time().

В параметре format распознаются следующие символы (см. табл.10.1):

Таблица 10.1 − Символы параметра format

Символ в строке format	Описание	Пример возвращаемого значения
C	Дата в формате ISO 8601 (добавлено в PHP 5)	2004-02-12T15:19:21+00:00
D	День месяца, 2 цифры с ведущими нулями	от *01* до *31*
F	Полное наименование месяца, например January или March	от *January* до *December*
G	Часы в 12-часовом формате без ведущих нулей	От *1* до *12*
G	Часы в 24-часовом формате без ведущих нулей	От *0* до *23*
I	Минуты с ведущими нулями	*00* to *59*
I (заглавная i)	Признак летнего времени	*1*, если дата соответствует летнему времени, иначе *0* otherwise.
l (строчная L)	Полное наименование дня недели	От *Sunday* до *Saturday*
M	Сокращенное наименование месяца, 3 символа	От *Jan* до *Dec*
N	Порядковый номер месяца без ведущих нулей	От *1* до *12*
O	Разница с временем по Гринвичу в часах	Например: *+0200*
T	Временная зона на сервере	Примеры: *EST, MDT* ...
W	Порядковый номер недели года по ISO-8601, первый день недели - понедельник (добавлено в PHP 4.1.0)	Например: *42* (42-я неделя года)
Y	Порядковый номер года, 4 цифры	Примеры: *1999, 2003*
Z	Порядковый номер дня в году (нумерация с 0)	От *0* до *365*

Любые другие символы, встреченные в строке format , будут выведены в результирующую строку без изменений. *Z* всегда возвращает *0* при использовании gmdate().

Пример вывода даты в формате:

Wednesday 15th of January 2003 05:51:38 AM

echo date("l dS of F Y h:i:s A");

Функция mktime − возвращает метку времени для заданной даты:

int mktime ([int $hour [, int $minute [, int $second [, int $month [, int $day [, int $year [, int $is_dst]]]]]]])

Функция возвращает метку времени Unix, соответствующую дате и времени, заданным аргументами. Метка времени − это целое число, равное разнице в секундах между заданной датой и началом Эпохи Unix (The Unix Epoch, 1 января 1970 г).

Функция mktime() возвращает FALSE.

Аргументы могут быть опущены в порядке справа налево. В этом случае их значения по умолчанию равны соответствующим компонентам локальной даты/времени.

Аргумент is_dst может быть установлен в 1, если заданной дате соответствует летнее время, 0 (в противном случае), или -1 (значение по умолчанию), если неизвестно, действует ли летнее время на заданную дату. В последнем случае PHP пытается определить это самостоятельно, что может привести к неожиданному результату (который, тем не менее, не всегда будет неверным).

10.2 *Порядок выполнения работы*

1 Создать файл со следующими функциями:

а) функция вывода (возвращения) текущей даты в формате число-месяц-год;

б) функция вывода (возвращения) текущей даты и времени;

в) функция вывода (возвращения) даты вашего рождения;

г) функция вывода (возвращения) количества дней, оставшихся до вашего рождения;

д) функция, которая возвращает массив значений. В качестве значений должны быть количество полных лет, количество полных месяцев, недель, дней, часов и минут, прошедших с момента вашего рождения.

2 Создать второй файл, где будут глобально определены переменные (константы), задающие дату вашего рождения.

3 Модифицировать файл с вашей биографией так, чтобы в нём красиво выводилась информация по созданным функциям.

10.3 *Контрольные вопросы*

1 Что возвращает функция mktime?

2 Что возвращает функция date?

3 Чем отличаются функции require от include?

4 Чем отличаются функции require_once от require?

5 Какую из функций, по вашему мнению, лучше использовать: require или include?

11 Работа со строками и регулярными выражениями (2 ч)

Цель: приобретение практических навыков по работе со строками и их обработке в PHP с помощью регулярных выражений.

11.1 *Теоретические сведения*

Строка − это набор символов. 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.

Строка может быть определена тремя различными способами:

- одинарными кавычками;
- двойными кавычками;
- heredoc-синтаксисом.

Способ 1. 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 (обратный слеш), т. е. экранировать ее. Если обратная косая черта должна находиться перед одинарной кавычкой, либо в конце строки, вам необходимо продублировать ее. Обратите внимание: если вы попытаетесь экранировать любой другой символ, обратная косая черта также будет напечатана! Так что, как правило, нет необходимости экранировать саму обратную косую черту.

echo 'это простая строка';

Способ 2. *Двойные кавычки*

Если строка заключена в двойные кавычки ("), PHP распознает большее количество управляющих последовательностей для специальных символов (см. таблицу 11.1):

Таблица 11.1 − Управляющие последовательности

Последовательность	Значение
\n	новая строка (LF или 0x0A (10) в ASCII)
\\	обратная косая черта
\$	знак доллара
\"	двойная кавычка
[0-7]{1,3}	последовательность символов, соответствующая регулярному выражению, символ в восьмеричной системе счисления
\x[0-9A-Fa-f]{1,2}	последовательность символов, соответствующая регулярному выражению, символ в шестнадцатеричной системе счисления

Повторяем, если вы захотите минимизировать любой другой символ, обратная косая черта также будет напечатана.

Способ 3. *Heredoc-синтаксис*

Другой способ определения строк − это использование heredoc-синтаксиса ("<<<"). После <<< необходимо указать идентификатор, затем − строку, 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 PHP: содержать только буквенно-цифровые символы и знак подчеркивания, и начинаться с not-a-number (нецифры) или знака подчеркивания.

В PHP есть два оператора для работы со строками. Первый − оператор конкатенации ('.'), который возвращает объединение левого и правого аргумен-

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

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

Регулярное выражение − это шаблон, применяемый к заданному тексту слева направо. Большая часть символов сохраняет свое значение в шаблоне и означает совпадение с соответствующим символом. Типичный пример: шаблон The quick brown fox соответствует той части строки, которая идентична приведенной фразе.

Шаблон составляется из набора модификаторов. Некоторые из которых приведены в таблице 11.2.

В PHP существует несколько функций для работы с регулярными выражениями: ereg(), ereg_replace(), eregi(), ereg_replacei() и split().

Функции с суффиксом i представляют собой аналоги функций без этого суффикса, но в отличие от них, не чувствительны к регистру операндов.

Таблица 11.2 − Набор модификаторов

Модификатор	Пример применения модификатора
\ − Следующий символ является специальным. Так же применяется для указания символов, которые могут использоваться в качестве модификаторов.	\n − соответствует символу перевода строки * − символ «*», а * − модификатор
^ − Маркер начала строки.	^abc − строка, начинающаяся с «abc».
$ − Маркер конца строки.	abc$ − строка, заканчивающаяся на «abc».
* − Предыдущий символ встречается 0 или больше раз.	Шаблону w* соответствуют строки what, buka, agwt
+ − Предыдущий символ встречается 1 или больше раз.	Шаблону w+ соответствуют строки what, agwt. Строка buka уже не соответствует.
? − Предыдущий символ встречается 0 или 1 раз.	Шаблону w?r соответствуют строки ara, awra.
. − Соответствует любому символу, отличному от "\n".	

Рассмотрим функцию ereg(), синтаксис которой:

int ereg(string pattern, string string, array [regs]);

Пусть дан некоторый адрес maxx@mail.ru. Очевидно, что правдоподобный адрес должен иметь вид "слово@слово.слово". В терминах шаблонов про-

извольный символ обозначается знаком "." (мы не будем сейчас учитывать тот факт, что в адресах допустимы не все символы). В каждом слове должен быть, по крайней мере, один символ, таким образом, шаблон слова будет иметь вид ".+". Вспомним теперь, что "." − это модификатор, и для явного указания точки (в качестве символа) нужно писать "\.".

Таким образом, шаблон будет иметь вид ".+@.+\..+".

Наша проверка будет иметь следующий вид:

```
if (ereg(".+@.+\..+", $email)) {
   echo "Адрес,  правильный";
   }
else {
   echo "Введите, адрес заново";
   }
```

После такой проверки мы можем быть уверены, что e-mail адрес имеет вид "слово@слово.слово".

11.2 *Порядок выполнения работы*

Создать страницу, где информация вводится в форму. Если информация не корректна, форма должна показываться заново, причём ошибочные места должны быть показаны в соответсвии со следующими пунктами:

1 Должен вводиться и проверяться электронный адрес (см. проверку в тексте выше)

2 Должен вводиться и проверяться телефон в формате (xxx) xxx-xx-xx.

3 Должен вводиться пароль, причём его «сила» должна проверяться на стороне сервера: он должен содержать минимум 2 цифры, минимум одну букву в верхнем регистре и одну букву в нижнем регистре. Он не должен содержать пробелов и минимальная длина пароля − 8 символов.

Если все введенные данные корректны, должна показываться страничка с надписью «Спасибо».

11.3 *Контрольные вопросы*

1 Что такое регулярное выражение?

2 В чём преимущество регулярных выражений в области проверки значений по сравнению с классическим подходом?

3 В чём ограничения регулярных выражений?

4 Перечислите основные способы задания строк.

5 Какими способами можно «склеить» две строки в PHP?

12 Работа с файлами (2 ч)

Цель: приобретение практических навыков работы с файлами и директориями в PHP.

12.1 *Теоретические сведения*

Последовательность работы с файлом в PHP, как и в C, стандартная:

1 Открыть файл (создать, если не его существует).

2 Осуществить требуемые манипуляции с файлом (чтение, запись).

3 Закрыть файл.

Для открытия файлов существует функция fopen:

resource fopen (string $filename , string $mode [, bool $use_include_path [, resource $zcontext]])

fopen() закрепляет именованный ресурс, указанный в аргументе filename , за потоком. Если filename передан в форме "scheme://...", он считается URL и PHP проведёт поиск обработчика протокола (также известного как «обвёртка») для этой схемы. Если ни одна обвёртка не закреплена за протоколом, PHP выдаст замечание, чтобы помочь вам отследить потенциальную проблему в вашем скрипте, и продолжит выполнение, как будто filename указывает на обыкновенный файл.

Если PHP решил, что filename указывает на локальный файл, тогда он попытается открыть поток к этому файлу. Файл должен быть доступен PHP, так что вам следует убедиться, что права доступа на файл разрешают это. Если включен безопасный режим или open_basedir, накладываются дальнейшие ограничения.

Параметр mode указывает тип доступа, который вы запрашиваете у потока. Его описано в таблице12.1.

Таблица 12.1 − Список возможных режимов для fopen() с использованием mode:

mode	Описание
'r'	Открывает файл только для чтения; помещает указатель в начало файла.
'r+'	Открывает файл для чтения и записи; помещает указатель в начало файла.
'w'	Открывает файл только для записи; помещает указатель в начало файла и обрезает файл до нулевой длины. Если файл не существует - пробует его создать.
'w+'	Открывает файл для чтения и записи; помещает указатель в начало файла и обрезает файл до нулевой длины. Если файл не существует - пробует его создать.
'a'	Открывает файл только для записи; помещает указатель в конец файла. Если файл не существует - пытается его создать.
'a+'	Открывает файл для чтения и записи; помещает указатель в конец файла. Если файл не существует - пытается его создать.

Необязательный третий параметр use_include_path может быть установлен в '1' или TRUE, если вы также хотите провести поиск файла в include_path.

Если открыть файл не удалось, функция вернёт FALSE и сгенерирует ошибку уровня E_WARNING. Вы можете использовать @ для того, чтобы подавить это предупреждение.

На платформе Windows необходимо не забывать экранировать все обратные слеши(\) в пути к файлу или использовать прямые слеши (/).

```
$handle = fopen("c:\\data\\info.txt", "r");
```

Для записи в файл существует функция fwrite:

```
int fwrite  ( resource $handle  , string $string  [, int $length  ] )
```

fwrite() записывает содержимое string в файловый поток handle . Если передан аргумент length , запись остановится после того, как length байтов будут записаны или будет достигнут конец строки string , смотря, что произойдёт первым.

fwrite() возвращает количество записанных байтов или FALSE в случае ошибки.

Для чтения из файла существует функция fread:

```
string fread  ( resource $handle  , int $length  )
```

fread() читает до length байтов из файлового указателя handle . Чтение останавливается при достижении length байтов, EOF (конца файла) или (для сетевых потоков) когда пакет становится доступным, что бы ни произошло первым.

Для получения размера файла существует функция filesize, принимающая в качестве аргумента имя файла. Пример чтения содержимого файла:

```
$filename = "/usr/local/something.txt";
$handle = fopen($filename, "r");
$contents = fread($handle, filesize($filename));
fclose($handle);
```

Для работы с директориями существуют функции is_dir, opendir, readdir, closedir. Функция is_dir осуществляет проверку того, является ли переданное имя файла директорией. Функция opendir возвращает ресурс открытой директории, через который впоследствии можно осуществлять вычитку записей из директории (файлов) с помощью функции readdir. После завершения работы с ресурсом директории его следует закрыть. Пример чтения директории /tmp и вывода содержащихся в ней файлов:

```
$dir = "/tmp/";
// Открыть заведомо существующий каталог и начать считывать его содержимое
if (is_dir($dir)) {
   if ($dh = opendir($dir)) {
      while (($file = readdir($dh)) !== false) {
         print "Файл: $file : тип: " . filetype($dir . $file) . "\n";
      }
      closedir($dh);
   }
}
```

Для удаления файла существует функция unlink, принимающая в качестве аргумента имя файла.

12.2 *Порядок выполнения работы*

Создать обозреватель удалённой файловой системы, отображающий в виде таблицы содержимое директории. В каждой строке обозревателя должно содержаться имя файла и действие. Если файл является директорией, то одно из действий − «войти в директорию», т. е. осуществить обзор этой директории. Если же это обычный файл, то должны присутствовать действия «удалить» и «просмотреть». При просмотре файла его содержимое должно выводиться в окно браузера. При обзоре директории должна быть предусмотрена кнопка (ссылка) перехода к вышестоящему (родительскому) каталогу.

12.3 *Контрольные вопросы*

1 Перечислите основные функции работы с файлами.
2 Каков должен быть порядок работы с файлами?
3 Какое значение возвращает функция fopen при невозможности открытия файла?
4 Перечислите основные функции для работы с директориями.
5 Что такое ресурс PHP?

13 Работа с базой данных (6 ч)

Цель: приобретение практических навыков работы с MySQL с помощью программы phpmyadmin и умений по программированию базы данных MySQL из PHP.

13.1 *Теоретические сведения*

Для создания таблиц базы данных нецелесообразно вводить запросы на создание таблиц напрямую. Лучше всего воспользоваться какой-либо графической программой, например, phpmyadmin.

Введение в SQL

Обычно каждая сущность должна состоять из таблицы. Таблица − это описание сущности на языке баз данных. Например, для описания автомобилей можно создать таблицу, состоящую из марки автомобиля, производителя и цены продажи.

Таблица состоит из колонок и строк. Колонки описывают свойства сущности (марка, производитель, цена автомобиля). Строки задают конкретные экземпляры сущности. Например, для автомобилей это может быть строка вида «Audi-Audi Inc. − 5000$».

Данные в таблицах хранятся неупорядоченно. Для того, чтобы однозначно идентифицировать каждую строку таблицы, нужно, чтобы у каждой строки был уникальный идентификатор, который на языке баз данных называется первичным ключом. В простейшем случае первичный ключ состоит из одной колонки. Первичный ключ может быть натуральным (естественным) или искусственным. Если существует такая возможность, лучше воспользоваться нату-

ральным первичным ключом (например, для автомобилей это может быть их марка). В общем случае натуральные ключи встречаются нечасто и поэтому вводят искусственные − для сущности это обычно числовой идентификатор, не имеющий отношения к сущности.

В phpmyadmin при создании сущностей можно задать колонку ID с типом INTEGER и флажком AUTO_INCREMENT, что автоматически увеличивает счётчик идентификаторов. Эта колонка и будет первичным ключом.

Допустим, в базе данных существует несколько сущностей: автомобили (automobile) с колонками id, seria, pruducer, year,cost и покупатели (customer) с колонками id, firstname, secondname.

Вставить данные (строку) в таблицу можно с помощью команды INSERT:

INSERT INTO automobile (seria, producer, year, cost) VALUES('Ford Scorpion 5M', 'Ford', 1999, 5000);

seria, producer, year, cost − это названия колонок, а 'Ford Scorpion 5M', 'Ford', 1999, 5000 − это соответствующие им значения в строке. Поле идентификатора (id) опущено, т. к. предполагается, что поле имеет флажок AUTO_INCREMENT, т.е. этому полю будет присвоено значение счётчика идентификаторов для таблицы, после чего счётчик будет увеличен.

Простейшая выборка осуществляется с помощью SQL-комманды SELECT:

SELECT * FROM automobile;

* − обозначает выборку всех колонок. При желании можно вручную задать имена колонок. Так как в данном случае не было задано условия выборки строк, то будет осуществлена выборка **всех** строк. Выборку можно ограничить конструкцией WHERE. Например, выберем все марки автомобилей и их цены, причём автомобиль должен быть произведен до 2000 года и иметь цену менее 4000:

SELECT seria, cost FROM automobile WHERE cost <= 4000 AND year <=2000;

Бывают следующие типы связей между сущностями: один ко многим («1−N»), один к одному («1−1») и многие ко многим («N−M»). Связи «один к одному» и «один ко многим» могут реализовываться следующим образом: в целевой таблице создаётся дополнительное поле, которое ссылается на первичный ключ другой таблицы. Это поле называется вторичным ключом. Например, если покупатель может купить только 1 автомобиль, то в таблице customer создаётся колонка automobile_id указывающая на приобретенный покупателем автомобиль. Для задания отношения «многие ко многим» нужно создать специальную таблицу-развязку, в которой будут 2 внешних ключа, указывающих на соответствующие сущности. Например, для нашего случая можно создать таблицу automobile_customer с полями (id, customer_id, automobile_id). Поле id необходимо, если покупатель может заказать два автомобиля одинаковой марки. Если такого ограничения нет, то уникальность строки гарантируется парой

customer_id, automobile_id, которая может выступать в качестве первичного ключа.

Можно осуществить выборку сразу из нескольких таблиц: для этого достаточно перечислить их в условии FROM. По умолчанию будет осуществлено декартово произведение строк таблиц, в чем обычно нет необходимости. Поэтому можно ограничить выборку только требуемыми строками. Например, следующий запрос выдаёт строки, состоящие из фамилии покупателя и приобретенного им автомобиля:

SELECT customer.second_name, automobile.serie FROM customer, automobile, automobile_customer WHERE automobile_customer.customer_id = customer.id AND automobile_customer.automobile_id = automobile.id

Сортировка осуществляется с помощью ключевого слова ORDER BY после секции WHERE, где через запятую перечисляются колонки, по которым нужно осуществлять сортировку.

Удалить строки из таблицы можно с помощью команды DELETE. Если Условие WHERE опущено, то будет осуществлено удаление всех записей из таблицы. Например, удалим все автомобили, которые были выпущены до 1950 года:

DELETE FROM automobile WHERE year < 1950.

PHP и SQL

Перед началом работы с MySQL из PHP нужно сначала установить соединение с базой данных:

```
$link = mysqli_connect('localhost', 'user', 'password', 'autos');
```

В данном случае localhost − это имя (или IP-адрес) компьютера с СУБД; user — имя пользователя базы данных; password − его пароль; autos − имя базы данных. Если произошла ошибка подключения, возвращается значение FALSE. В противном случае возвращается ресурс подключения.

Исполнять SQL-команды можно с помощью функции mysqi_query, которая принимает в качестве аргументов ресурс подключения и строку с запросом. В случае успеха функция возвращает ресурс результата, из которого можно выбрать данные с помощью функции mysqli_fetch_row. Например:

```
$result = mysqli_query($link, 'SELECT serie, year FROM automobiles');
if($result){
        while( $row = mysqli_fetch_row($result){
                printf('%s - %s',$row[0], $row[1]);
        }
}
```

Функция mysqli_fetch_row итерирует по ресурсу результата. Если все строки результата пройдены, то функция возвращает значение 0, иначе − массив, в котором хранятся значения для одной строки результата. Этот массив может индексироваться, т. е. в нашем примере первым элементом массива будет серия автомобиля, а вторым − год выпуска.

13.2 *Порядок выполнения работы*

Создать приложение для автоматизации работы приёмной комиссии ВУЗа.

1 Обеспечить возможность ввода специальностей и количества бюджетных мест по каждой из них.

2 Обеспечить возможность ввода дисциплин, по которым абитуриенты сдают вступительные экзамены.

3 Обеспечить возможность «связывания» специальности и дисциплин, по которым сдаются экзамены.

4 Обеспечить возможность ввода личных данных абитуриентов, с привязкой абитуриента к конкретной специальности и вводом баллов, которые студент набрал по дисциплинам.

5 Предоставить отчёты:

а) по каждой специальности вывести списки поступивших абитуриентов и не поступивших на бюджет абитуриентов;

б) по каждой дисциплине вывести упорядоченный список студентов, получивших по данной дисциплине наиболее высокие показатели.

13.3 *Контрольные вопросы*

1 Для чего предназначены реляционные базы данных?

2 Что такое первичный и вторичный ключи?

3 Как осуществлять выборку данных из нескольких таблиц?

4 Какие в PHP существуют функции для работы с MySQL?

Литература

1 Бенкен, Е. PHP, Mysql, XML: программирование для интернета/ Е. Бенкен − СПб.: БХВ-Петербург, 2007. − 336 с.

2 Мейер, Э. CSS-каскадные таблицы стилей. Подробное руководство, 2-е издание/ Э. Мейер. − Пер. с англ. СПб.: Символ-Плюс, 2007. − 576 с., ил.

3 Муссиано, Ч. HTML и XHTML. Подробное руководство / Ч. Муссиано, Б.Кеннеди. − Пер. с англ. − СПб.: Символ-Плюс, 2008. − 752 с., ил.

4 Антоновский, А.И. Психологические последствия Интернета/ А.И. Антоновский − М.: Изд. Центр «Академия», 2007. − 182 с.

5 Косарев, С.А. Интернет и его психология сегодня/ С. А. Косарев. − М.: Изд. Центр «Академия», 2007. − 79 с.

6 Шупейко, И.Г. Психология восприятия и переработки информации : лаб. практикум для студ. спец.I-58 01 01 «Инженерно-психологические обеспечение информационных технологий» днев. формы обуч. / И.Г. Шупейко. −. Минск : БГУИР, 2008. −79 с.