МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ АГЕНТСТВО ЖЕЛЕЗНОДОРОЖНОГО ТРАНСПОРТА
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
САМАРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ПУТЕЙ СООБЩЕНИЯ

Приложение 2
к рабочей программе дисциплины

ОЦЕНОЧНЫЕ МАТЕРИАЛЫ ДЛЯ ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ ПО ДИСЦИПЛИНЕ (МОДУЛЮ)

Программирование сетевых задач

(наименование дисциплины(модуля)

Направление подготовки / специальность

09.03.03 Прикладная информатика
(код и наименование)

Направленность (профиль)/специализация

Прикладная информатика на железнодорожном транспорте
(наименование)

Содержание

1. Пояснительная записка

Цель промежуточной аттестации – оценивание промежуточных и окончательных результатов обучения по дисциплине, обеспечивающих достижение планируемых результатов освоения образовательной программы.

Перечень компетенций, формируемых в процессе освоения дисциплины

Код и наименование компетенции
ОПК-7 Способен разрабатывать алгоритмы и программы, пригодные для практического применения;
ОПК-7.2 Занимается веб-разработкой, в том числе с использованием скриптовых языков программирования
ПК-1 Способен разрабатывать требования и проектировать программное обеспечение
ПК-1.2 Разрабатывает программный код и осуществляет отладку на языках программирования высокого уровня

Результаты обучения по дисциплине, соотнесенные с планируемыми результатами освоения образовательной программы

Код и наименование индикатора достижения компетенции	Результаты обучения по дисциплине	Оценочные материалы
ОПК-7.2 Занимается веб-разработкой, в том числе с использованием скриптовых языков программирования	Обучающийся знает: теорию сетевых алгоритмов	Задания (тест №1 - №5)
	Обучающийся умеет: переводить постановку задач в сетевые структуры	Задания 2
	Обучающийся владеет: приемами составления алгоритмов решения сетевых задач	Задания 3
ПК-1.2 Разрабатывает программный код и осуществляет отладку на языках программирования высокого уровня	Обучающийся знает: преимущества и особенности программирования на языке высокого уровня, основные понятия, конструкции и структуры языка программирования для решения задач.	Задания (тест №6 - №10)
	Обучающийся умеет: работать с современными средствами программирования на языках высокого уровня.	Задания 4
	Обучающийся владеет: инструментальными средствами, методами и навыками разработки программного обеспечения с использованием языка программирование высокого уровня.	Задания 5

Промежуточная аттестация (экзамен) проводится в одной из следующих форм:
1) ответ на билет, состоящий из теоретических вопросов и практических заданий;
2) выполнение заданий в ЭИОС СамГУПС.

2. Типовые[1] контрольные задания или иные материалы для оценки знаний, умений, навыков и (или) опыта деятельности, характеризующих уровень сформированности компетенций

2.1 Типовые вопросы (тестовые задания) для оценки знаниевого образовательного результата

Проверяемый образовательный результат

Код и наименование индикатора достижения компетенции	Образовательный результат
ОПК-7.2.1	Обучающийся знает: теорию сетевых алгоритмов
1. Java. класс в Java для работы с сокетами дейтаграммного типа + ***DatagramSocket*** - URL - InetAddress - BufferedReader - BufferedWriter 2. Java. классв Java дляработыс URL + ***URL*** - DatagramSocket - InetAddress - BufferedReader - BufferedWriter 3. Java. метод getlnputStream() + ***возвращает входной поток типа InputStream*** - возвращает выходной поток типа OutputStream - возвращает длину полученной информации в байтах - возвращает полученную информацию в виде объекта типа Object - ничего не возвращает 4. Java. метод getOutputStream() + ***возвращает входной поток типа OutputStream*** - возвращает выходной поток типа InputStream - возвращает длину полученной информации в байтах - возвращает полученную информацию в виде объекта типа Object - ничего не возвращает 5. Какая технология в Java позволяет заменить CGI программы, находящиеся на сервере? + ***Сервлеты*** - Апплеты - Многопоточность - Синхронизация - В Java нет такой технологии	
ПК-1.2.1	Обучающийся знает: преимущества и особенности программирования на языке высокого уровня, основные понятия, конструкции и структуры языка программирования для решения задач.
6. На каком языке можно написать CGI программу? + ***Все ответы верны***	

[1]Приводятся типовые вопросы и задания. Оценочные средства, предназначенные для проведения аттестационного мероприятия, хранятся на кафедре в достаточном для проведения оценочных процедур количестве вариантов. Оценочные средства подлежат актуализации с учетом развития науки, образования, культуры, экономики, техники, технологий и социальной сферы. Ответственность за нераспространение содержания оценочных средств среди обучающихся университета несут заведующий кафедрой и преподаватель – разработчик оценочных средств.

- Python
- Pascal
- Perl, PHP
- C, C++

7. Java. Пакет в Java, в котором множество классов связанных для работы с сетью
+ ***java.net***
- javafx
- java.util
- javax.swing
- java.awt

8. Java. Объект какого класса нужно создать, чтобы узнать к какому типу относится переданный файл?
+ ***URLConnection***
- Iterator
- StringTokenizer
- Vector
- ArrayList

9. Java. Класспредставляющийсокетв Java
+ ***Socket***
- SocketClass
- SocketURL
- SocketConnection

10. Java. К какому пакету относится класс Socket?
+ ***java.io***
- javax.swing
- java.awt
- javafx
- java.util

2.2 Типовые задания для оценки навыкового образовательного результата

Проверяемый образовательный результат

Код и наименование индикатора достижения компетенции	Образовательный результат
ОПК-7.2.2	Обучающийся умеет: переводить постановку задач в сетевые структуры

ЯЗЫК РАЗМЕТКИ ГИПЕР ТЕКСТОВ HTML

Цель работы: приобретение навыков создания HTML – документов.

Изучаемые вопросы

1. Базовый синтаксис языка, основные элементы HTML–документа.
2. Заголовочная часть документа <HEAD>.
3. Комментарии.
4. Тэги тела документа.
5. Ссылки в HTML-документе.
6. Графика внутри HTML-документа.
7. Карты сообщений.
8. HTML фреймы.

Постановка задачи

Создайте HTML - документ, состоящий из двух горизонтальныхфрэймов.Вправомфрэймеразместитеизображение.Понажатиюнаотдельныеэлементыиз ображениявовторомфрэймедолжнопоявитьсяописание данного элемента.

Теоретические сведения

Базовый синтаксис языка, основные элементы HTML–документа

HTML (от англ. Hyper Text Markup Language — «язык разметки гипертекста») — стандартный язык разметки документов во Все-мирной паутине. Большинство веб-страниц создаются при помощи языка HTML (или XHTML). Язык HTML интерпретируется браузе-рами и отображается в виде документа в удобной для пользователя форме. Когда документ создан с использованием HTML,WEB-браузер может интерпретировать HTML для выделения различных элементов документа и первичной их обработки. ИспользованиеHTML позволяет форматировать документы для их представления с использованием шрифтов, линий и других графических элементов на любой системе, их просматривающей.

HTML-тэгимогутбытьусловноразделенына две категории:

1. тэги,определяющие,какбудетотображатьсяWEB-браузеромтелодокументавцелом;
2. тэги, описывающие общие свойства документа, такие как за-головокилиавтор документа.

HTML-документы могут быть созданы при помощи любого тек-стовогоредактораилиспециализированныхHTML-редакторовиконвертеров.

Все тэги HTML начинаются с "<" (левой угловой скобки) и за-канчиваются символом ">" (правой угловой скобки). Как правило,существует стартовый тэг и завершающий тэг. Завершающий тэготличается от стартового прямым слешем перед текстом внутри уг-ловых скобок. В примере тэг <TITLE> говорит WEB-браузеру обиспользовании формата заголовка, а тэг </TITLE> - о завершениитекста заголовка.

ДокументвформатеHTMLсостоитизтрехчастей:

1. строки,содержащейинформациюоверсииHTML;
2. разделазаголовков(определяемогоэлементомHEAD);
3. тела, которое включает собственно содержимое документа.Теломожетводитьсяэлементом BODY илиэлементомFRAMESET.

Перед каждым элементом или после каждого элемента можетнаходиться пустое пространство (пробелы, переход на новую стро-ку, табуляции и комментарии). Разделы 2 и 3 должны отделятьсяэлементомHTML.

ВотпримерпростогодокументаHTML:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<HTML>
<HEAD>
<TITLE>Документ HTML</TITLE>
</HEAD>
<BODY>
<P>Первый HTML документ.
6

</BODY>
</HTML>
```

Заголовочная часть документа <HEAD>

Тэг заголовочной части документа должен быть использован сразу после тэга <HTML> и более нигде в теле документа. Данный тэг представляет из себя общее описание документа.
Стартовый тэг

<HEAD> помещается непосредственно перед тэгом <TITLE> и дру- гими тэгами, описывающими документ, а завершающий тэг

</HEAD> размещается сразу после окончания описания документа. Например:

<HTML>

<HEAD>

<TITLE>Списокучащихся</TITLE>

</HEAD>

...

Комментарии

HTML позволяет вставлять в тело документа комментарии, ко- торые сохраняются при передаче документа по сети, но не отобра- жаются браузером. Синтаксис комментария:

<!-- Это комментарий -->

Комментарии могут встречаться в документе где угодно и в лю- бом количестве.

Тэги тела документа

Тело документа должно находиться между тэгами <BODY> и

</BODY>. Это та часть документа, которая отображается как тек- стовая и графическая (смысловая) информация вашего документа.

Ссылки в HTML-документе

Для того, чтобы браузер отобразил ссылку на URL, необходимо выделить URL специальными тэгами в HTML-документе. Синтак- сис HTML, позволяющий это сделать - следующий:

<A HREF="URL"> текст подсвечивается как ссылка </A>

Тэг <A HREF="URL">открывает описание ссылки, а тэг </A> - закрывает его. Любой текст, находящийся между данными двумя тэгами подсвечивается специальным образом Web-браузером.

Графика внутри HTML-документа

Существует возможность включения ссылок на графические и иные типы данных в HTML-документах. Делается это при помощи тэга <IMG...ISMAP>.

Синтаксис тэга

<IMG SRC="URL" ALT="text" HEIGHT=n1 WIDTH=n2

ALIGN=top|middle|bottom|texttop ISMAP>

Опишем элементы синтаксиса тэга:

URL – Обязательный параметр, имеющий такой же синтаксис, как и стандартный URL. Данный URL указывает браузеру где нахо- дится рисунок. Рисунок должен храниться в графическом формате, поддерживаемом браузером.

ALT="text" – Данный необязательный элемент задает текст, ко- торый будет отображен браузером, не поддерживающим отображе- ние графики или с отключенной подкачкой изображений.

HEIGTH=n1 – Данный необязательный параметр используется для указания высоты рисунка в пикселях. Если данный параметр не указан, то используется оригинальная высота рисунка.

WIDTH=n2 – Параметр также необязателен, как и предыдущий.

Позволяет задать абсолютную ширину рисунка в пикселях.

ALIGN – Данный параметр используется, чтобы сообщить брау- зеру, куда поместить следующий блок текста. Это позволяет более строго задать расположение элементов на экране. Если данный параметр не используется, то большинство браузеров располагает изображение в левой части экрана, а текст справа от него.

ISMAP – Этот параметр сообщает браузеру, что данное изобра- жение позволяет пользователю выполнять какие-либо действия, щелкая мышью на определенном месте изображения.

Пример:

<IMG SRC="http://www.bntu.by/images/povt.jpg" ALT="ПОВТиАСлого" ALIGN="top" ISMAP>

Карты сообщений

Создание карты изображения является одной из привлекатель- нейших возможностей HTML, позволяющей пользователю привя- зывать ссылки на другие документы к отдельным частям изображе- ний. Щелкая мышью на отдельных частях изображения, пользова- тель может выполнять те или иные действия, переходить по той или иной ссылке на другие документы и т.п.

Чтобы включить поддержку карты для изображения, необходимо ввести дополнительный параметр в тэг IMG:

<IMG SRC="url" USEMAP="url#map_name" >

Синтаксис:

<MAP NAME="map_name"> - Данный тэг определяет начало описания карты с именем map_name.

<AREA> - Описывает участок изображения и ставит ему в соот- ветствие URL.

Параметры:

SHAPE - Необязательный параметр, указывающий на форму оп- ределяемой области изображения. Может принимать значения:

default - по умолчанию (обычно прямоугольник) rect - прямоугольник

circle - круг

poly - многоугольник произвольной формы

COORDS - Координаты в пикселях описываемой области. Для прямоугольника это четыре координаты левого верхнего и правого

нижнего углов, для круга - три координаты (две - центр круга, тре- тья - радиус). Для многоугольника это описание каждого угла в двух координатах - соответственно число координат равно удвоен- ному количеству углов.

Координаты считаются с нуля, поэтому для описания области

100 на 100 используется описание:

<AREA COORDS="0,0,99,99" >

HREF="url" - Описание ссылки, действия по которой будут вы- полняться при щелчке мыши в заданной области.

NOHREF - Параметр, указывающий, что ссылка отсутствует для данного участка. По умолчанию, если не указан параметр HREF, то считается что действует параметр NOHREF. Также, для всех неопи- санных участков изображения считается, что используется пара- метр

NOHREF.

Если две описанных области накладываются друг на друга, то используется ссылка, принадлежащая первой из описанных облас- тей.

</MAP> - Завешающий тег Пример:

<MAP NAME="map_name">

<AREA [SHAPE=" shape "] COORDS="x,y,..."

[HREF=" reference "] [NOHREF]>

</MAP>

HTMLфреймы

Фреймы, позволяющие разбивать Web-страницы на множест- венные скроллируемые подокна, могут значительно улучшить внешний вид и функциональность Web-приложений. Каждое по- докно, или фрейм, может иметь следующие свойства:

Каждый фрейм имеет свой URL, что позволяет загружать его не- зависимо от других фреймов

Каждый фрейм имеет собственное имя (параметр NAME), по- зволяющее переходить к нему из другого фрейма

Размер фрейма может быть изменен пользователем прямо на эк- ране при помощи мыши (если это не запрещено указанием специ- ального параметра)

Формат документа, использующего фреймы, внешне очень на- поминает формат обычного документа, только вместо тэга BODY используется контейнер FRAMESET, содержащий описание внут- ренних HTML-документов, содержащий собственно информацию, размещаемую во фреймах.

<FRAMESET COLS="value" | ROWS="value">

<FRAME SRC="url1">

<FRAME ...>

...

</FRAMESET>

FRAMESET

<FRAMESET [COLS="value" | ROWS="value"]>

Тэг<FRAMESET>имеетзавершающийтэг</FRAMESET>. Все, что может находиться между этими двумя тэгами, это тэг

<FRAME>, вложенные тэги <FRAMESET> и </FRAMESET>, а также контейнер из тэгов <NOFRAME> и </NOFRAME>, который позволяет строить двойные документы для браузеров, поддержи- вающих фреймы и не поддерживающих фреймы.

Данный тэг имеет два взаимоисключающих параметра: ROWS и

COLS.

ROWS="список-определений-горизонтальных-подокон" - Дан- ный тэг содержит описания некоторого количества подокон, разде- ленные запятыми. Каждое описание представляет собой числовое значение размера подокна в пикселях, процентах от всего размера окна или связанное

масштабное значение. Отсутствие атрибута ROWS определяет один фрейм, величиной во все окно браузера.

Синтаксис используемых видов описания величин подокон: value - Простое числовое значение определяет фиксированную

высоту подокна в пикселях.

value% - Значение величины подокна в процентах от 1 до 100. Если общая сумма процентов описываемых подокон превышает 100, то размеры всех фреймов пропорционально уменьшаются до суммы 100%. Если, соответственно, сумма меньше 100, то размеры пропорционально увеличиваются.

value* - Вообще говоря, значение value в данном описании явля- ется необязательным. Символ "*" указывает на то, что все остав- шееся место будет принадлежать данному фрейму.

COLS="список-определений-горизонтальных-подокон" - То же самое, что и ROWS, но делит окно по вертикали, а не по горизонта- ли.

Пример:

<FRAMESET COLS="50,*,50"> - описывает три фрейма, два по 50 точек справа и слева, и один внутри этих полосок.

FRAME

<FRAME SRC="url" [NAME="frame_name"] [MARGINWIDTH="nw"] [MARGINHEIGHT="nh"]

[SCROLLING=yes|no|auto] [NORESIZE]>

Данный тэг определяет фрейм внутри контейнера FRAMESET. SRC="url" - Описывает URL документа, который будет отобра-

жен внутри данного фрейма. Если он отсутствует, то будет отобра- жен пустой фрейм.

NAME="frame_name" - Данный параметр описывает имя фрейма. Имя фрейма может быть использовано для определения действия с данным фреймом из другого HTML-документа или фрейма (как правило, из соседнего фрейма этого же документа). Имя обязатель- но должно начинаться с символа. Содержимое поименованных фреймов может быть задействовано из других документов при по- мощи специального атрибута TARGET, описываемого ниже.

MARGINWIDTH="value" - Это атрибут может быть использован, если автор документа хочет указать величину разделительных по- лос между фреймами сбоку. Значение value указывается в пикселях и не может быть меньше единицы.

MARGINHEIGHT="value" - То же самое, что и MARGINWIDTH,

но для верхних и нижних величин разделительных полос.

SCROLLING="yes | no | auto" - Этот атрибут позволяет задавать наличие полос прокрутки у фрейма.

NORESIZE - Данный атрибут позволяет создавать фреймы без возможности изменения размеров.

NOFRAMES - Данный тэг используется в случае, если создается документ, который может просматриваться как браузерами, под- держивающими фреймы, так и браузерами, их не поддерживающи- ми. Данный тэг размещается внутри контейнера FRAMESET, а все, что находится внутри тэгов <NOFRAMES> и </NOFRAMES> игно- рируется браузерами,

поддерживающими фреймы.

Примеры:

```
<FRAMESET ROWS="*,*">
<NOFRAMES>
<H1>Ваша версия WEB-браузера не поддерживает фрей- мы!</H1>
</NOFRAMES>
<FRAMESET COLS="65%,35%">
<FRAME SRC="link1.php">
<FRAME SRC="link2.php">
</FRAMESET>
<FRAMESET COLS="*,40%,*">
<FRAME SRC="link3.php">
<FRAME SRC="link4.php">
<FRAME SRC="link5.php">
</FRAMESET>

</FRAMESET>
```

Контрольные вопросы:

1. На какие части разделяется HTML-документ?
2. При помощи какого тэга в HTML-документ добавляется графика?
3. Назовите основные тэги формы?
4. Для чего используется карта сообщений?
5. Для чего используется фрэйм NOFRAMES?

ОПК-7.2.3	Обучающийся владеет: приемами составления алгоритмов решения сетевых задач

КАСКАДНЫЕ ТАБЛИЦЫ СТИЛЕЙ (CSS)

Цель работы: приобретение навыков создания HTML – документов с использованием CSS.

Изучаемые вопросы

1. Синтаксис и элементы CSS.
2. Добавление стилей CSS в HTML-документ.
3. Связывание.
4. Внедрение.
5. Встраивание в теги документа.
6. Импортирование.

Постановка задачи

Создать HTML-документ и оформить его при помощи CSS.

Все стили необходимо вынести в отдельный файл.

Теоретические сведения

CSS (англ. Cascading Style Sheets - каскадные таблицы стилей) — формальный язык описания внешнего вида документа, написанного с использованием языка разметки. Преимущественно используется как средство описания, оформления внешнего вида веб-страниц, написанных с помощью языков разметки HTML и XHTML, но мо- жет также применяться к любым XML-документам, например, к SVG или XUL.

Основным понятием CSS является стиль – т. е. набор правил оформления и форматирования, который может быть применен к различным элементам документа. В стандартном HTML для при- своения какому-либо элементу определенных свойств (таких, как цвет, размер, положение на странице и т. п.) приходилось каждый раз описывать эти свойства, увеличивая размер файла и время за- грузки на компьютер просматривающего ее пользователя.

CSS действует более удобным и экономичным способом. Для присвоения какому-либо элементу определенных характеристик необходимо один раз описать этот элемент и определить это описание как стиль, а в дальнейшем просто указывать, что элемент, который нужно оформить соответствующим образом, должен принять свойства указанного стиля.

Более того, можно сохранить описание стиля не в тексте кода документа, а в отдельном файле – это позволит использовать описание стиля на любом количестве Web-страниц, а также изменить оформление любого количества страниц, исправив лишь описание стиля в одном (отдельном) файле.

Кроме того, CSS позволяет работать со шрифтовым оформлением страниц на гораздо более высоком уровне, чем стандартный HTML, избегая излишнего утяжеления страниц графикой.

```
<html>

<head>

<style type="text/css">

.newfont{font-size:24px; color:#CC9933}

</style>

<title>Классы для создания тэгов.</title>

<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">

</head>

<body>

<blockquote class=" newfont">Заголовок</blockquote>

</body>

</html>
```

Данный пример иллюстрирует вариант объявления нового стиля в документе и потом его использования.

Синтаксис и элементы CSS

Добавление стилей CSS в HTML-документ

Существует несколько способов связывания документа и табли- цы стилей:

Связывание - позволяет использовать одну таблицу стилей для форматирования многих страниц HTML

Внедрение - позволяет задавать все правила таблицы стилей не- посредственно в самом документе

Встраивание в теги документа - позволяет изменять форматиро- вание конкретных элементов страницы

Импортирование - позволяет встраивать в документ таблицу стилей, расположенную на сервере.

Связывание

Напомним, что информация о стилях может располагаться либо в отдельном файле, либо непосредственно в коде документа. Распо- ложение описания стилей в отдельном файле целесообразно при применении стилей при количестве страниц более одной. Для этого необходимо создать текстовый файл, описать необходимые стили и в коде документов, которые будут использовать эти стили необхо- димо создать ссылку на данный файл. Отметим, что данный файл может располагаться где угодно, необходимым условием является только то, чтобы браузер клиента мог его загрузить на свою сторо- ну. Осуществляется это с помощью тега LINK, располагающегося внутри тега HEAD документов:

<LINK REL=STYLESHEET TYPE="text/css" HREF="URL">

Первые два параметра этого тега являются зарезервированными именами, требующимися для того, чтобы сообщить браузеру, что на этой страничке будет использоваться CSS. Третий параметр – HREF= «URL» – указывает на файл, который содержит описания стилей. Этот параметр должен содержать либо относительный путь к файлу – в случае, если он находится на том же сервере, что и до- кумент, из которого к нему обращаются – или полный URL («http://...») в случае, если файл стилей находится на другом серве- ре.

<head>

<title></title>

<meta http-equiv="content-type" cotent=

"text/html; charset=windows-1251">

<link rel="stylesheet" href="css/default.css">

</head>

Внедрение

Второй вариант, при котором описание стилей располагается в коде Web-странички, внутри тега HEAD, в теге <STYLE type="text/css">... </STYLE. В этом случае вы можете использовать эти стили для элементов, располагающихся в пределах странички. Параметр type="text/css" является обязательным и служит для ука- зания браузеру использовать CSS.

<head>

<style type="text/css" >

.el_cl_1{display:inline; z-index:1};

.el_lst{display:list-item; margin:-1%; background:#ff0000 url("bc.jpg") no-repeat};

</style>

<title></title>

<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">

</head>

Встраивание в теги документа

Данный, третий по счету, метод позволяет располагать описания стилей непосредственно внутри тега элемента, который описывает. Это осуществляется при помощи параметра STYLE, используемого при применении CSS с большинством стандартных тегов HTML. Данный метод нежелателен, т.к. приводит к потере одного из ос- новных преимуществ CSS – возможности разделения информации и описания оформления информации.

<blockquote style="color:#CCFF66">Внимание!</blockquote>

Импортирование

В теге <STYLE> можно импортировать внешнюю таблицу сти- лей с помощью свойства @import таблицы стилей:

@import: url(styles.css);

Его следует задавать в начале стилевого блока или связываемой таблицы стилей перед заданием остальных правил. Значение свой- ства @import является URL файла таблицы стилей.

Заметим, что импортирование от связывания отличается тем, что при импортировании можно не только поместить внешнюю таблицу стилей в документ, но и поместить одну внешнюю таблицу стилей в другую.

<head>

<title>Untitled </title>

<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />

<style type="text/css">

@import url('css/default.css');

</style>

</head>

Ниже приведен код примера.

<STYLEtype="text/css">

H4 {text-decoration: underline;} A {text-decoration: none;}

i {text-decoration:line-through;} b {text-decoration:overline;}

H5 {text-align: center}

b.cap {text-transform:capitalize;}

.otstup {text-indent: 50pt;}

.interval {line-height: 50 %}

</STYLE>

<h4>Подчеркнутыйзаголовок (underline)</h4>

Этоссылкабезподчер- кивания (none)

Перечеркнутый курсивный текст (line- through)

Текст с подчеркиванием сверху (overline)

каждая первая буковка слова боль- шая

<h5>Заголовок по центру</h5>

Абзац с отступом для лучшего вос- приятия текстов при оформлении ваших страничек

Текст с уменьшеным интервалом на

50% между строками.

Данным метод можно применять в декоративных це- лях

Увеличим проме- жутки между словами

Уменьшим про- межутки между буквами

Формуласпирта

C

<span style= vertical- align:sub;color:blue;>2

H

5

OH

Контрольные вопросы:

1. Для чего используются каскадные таблицы стилей?
2. Какими способами таблицы стилей связываются с элементами документа?
3. Каковы основные отличия импортирования от связывания?
4. Каким образом сделать так, чтобы изменялся цвет ссылок только внутри тэга ?

ПК-1.2.2	Обучающийся умеет: работать с современными средствами программирования на языках высокого уровня.

ЯЗЫК СОСТАВЛЕНИЯ СЦЕНАРИЕВ JavaScript.

Цель работы: приобретение навыков создания HTML –документов с использованием JavaScript.

Изучаемые вопросы

1. Типы данных
2. Преобразование типов данных
3. Объявление переменных

4. Оператор цикла

5. Объектная модель JavaScript

6. Функции и методы

Постановка задачи

Создать HTML-документ, содержащий массив, в котором хра- нятся ссылки на объекты.

Теоретические сведения

JavaScript - это язык для составления сценариев, позволяющих выполнять разные действия непосредственно на машине пользова- теля. Располагаются данные сценарии внутри HTML документов.

JavaScript применяется для проверки правильности заполнения форм, создания удобной навигации и т.д.

Это язык программирования, который понятен браузеру. Это оз- начает, что браузер умеет выполнять (интерпретировать) команды этого языка.

Программу на JavaScript можно помещать внутрь HTML-кода или держать в отдельном файле. Этот файл браузер прочитает (по специальной команде) во время интерпретации HTML-программы.

Программы на JavaScript (их называют скриптами) не работают самостоятельно. Коды JavaScript дополняют коды HTML и "живут" только вместе с ними. Даже если они расположены в отдельном файле

Размещение JavaScript на HTML-странице

Скрипт размещается между двумя парными тегами

<SCRIPT>...</SCRIPT>. Обычно запись скрипта выглядит так:

<SCRIPT

language=JavaScript>

<!--

...

КоднаJavaScript

...

//-->

</SCRIPT>

<NOSCRIPT>

...

Для браузеров, которыене поддерживают JavaScript

...

</NOSCRIPT>

Начало скрипта

Скрипт представлен как HTML-комментарий, чтобы не "смущать" браузеры, которые о

скриптах не знают.

Конец скрипта Эта команда -

специально для пользователей, у которых

браузер не понимает скриптов.

Типы данных

JavaScript распознает следующие типы величин:

– числа, типа 42 или 3.14159;

– логические (Булевы), значения true или false;

– строки, типа "Howdy!";

– пустой указатель, специальное ключевое слово, обозна- чающее нулевое значение.

Преобразование типов данных

Тип переменной зависит от того, какой тип информации в ней хранится. JavaScript не является жестко типизированным языком. Это означает, что программист может не определять тип данных переменной, в момент ее создания. Тип переменной присваивается переменной автоматически в течение выполнения. Таким образом можно определить переменную следующим способом:

var answer = 42

А позже, можно присвоить той же переменной, например сле- дующее значение:

answer = "Thanks for all the fish..."

Объявления переменных

Переменная должна быть объявлена до ее использования. Для объявления используется ключевое слово var:

var x; // переменная с именем "x".

var y = 5; // описание с присвоением числа.

var mes = "дядя Федор"; // описание с присвоением строки.

Оператор цикла

for (нач; усл; приращ) команда

Команда "нач" выполняется один раз перед входом в цикл.

Цикл состоит в повторении следующих действий: проверка условия "усл";

выполнение команды "команда";

выполнение команды "приращ".

Если условие ложно, цикл прекращается ("команда" и "приращ"

после отрицательной проверки не работают).

// Произведение нечетных чисел массива var set = new Array(1,2,3,4,5,6,7,8,9); var p = 1;

for(var i=0; i<set.length; i++) if (set[i]%2) p *= set[i];

alert(p);

Условная команда

if (условие) команда1; else команда2;

или

if (условие) команда1;

Если условие принимает значение true, выполняется команда1, иначе команда2. В сокращенной форме ветвь else отсутствует.

// Абсолютное значение числа

var x = -25.456; if (x < 0) x = -x;

alert(x);

Объектная модель JavaScript

JavaScript основан на простом объектно-ориентированном при- мере. Объект - это конструкция со свойствами, которые являются переменными JavaScript. Свойства могут быть другими объектами. Функции, связанные с объектом известны как методы объекта.

В дополнение к объектам, которые сформированы в Navigator client и LiveWire server, вы можете определять ваши собственные объекты.

Объекты и Свойства

Объект JavaScript имеет свойства ассоциированные с ним. Об- ращаться к свойствам объекта необходимо следующей простой сис- темой обозначений:

objectName.propertyName

И имя объекта и имя свойства чувствительны к регистру. Например, пусть существует объект, с именем myCar. Можно за-

дать свойства, именованные make и year следующим образом:

myCar.make = "Ford" myCar.year = 69;

Также можно обратиться к этим свойствам, используя систему обозначений таблицы следующим образом:

myCar["make"] = "Ford" myCar["year"] = 69;

Функции и Методы

Функции - один из фундаментальных встроенных блоков в JavaScript. Функция - JavaScript процедура - набор утверждений, которые выполняют определенную задачу.

Определение функции состоит из ключевого слова function , со- провождаемого

1. именем функции;

2. списком аргументов функции, приложенной в круглых скоб- ках, и отделяемые запятыми;

3. JavaScript утверждениями, которые определяют функцию, приложенные в фигурных

скобках, {...}.

Можно использовать любые функции, определенные в текущей странице. Лучше всего определять все функции в HEAD страницы. Когда пользователь загружает страницу, сначала загружаются функции.

Утверждения в функциях могут включать другие обращения к функции.

Например, есть функция с именем pretty_print:

function pretty_print(string)

{ document.write("" + string) }

Эта функция принимает строку как аргумент, прибавляет неко- торые теги HTML, используя оператор суммы (+), затем показывает результат в текущем документу.

Определение функции не выполняет ее. Для этого необходимо вызвать функцию, чтобы выполнить ее. Например, можновызыватьфункцию pretty_print следующимобразом:

< SCRIPT>

pretty_print("This is some text to display")

</ SCRIPT>

Аргументы функции сохраняются в таблице. Внутри функции, вы можете адресовать параметры следующим образом:

functionName.arguments [i]

Где functionName - имя функции, и i - порядковое число аргу- мента, начинающегося с нуля. Так, первый аргумент в функции, с именем myfunc, будет myfunc.arguments [0]. Общее число аргумен- тов обозначено переменным arguments.length.

Определение Методов

Метод - функция, связанная с объектом. Метод определяется та- ким же образом, как и стандартная функция. Затем, используется следующий синтаксис, чтобы связать функцию с существующим бъектом:

object.methodname = function_name

Где object - существующий объект, methodname - имя, которое присвоено методу, и function_name - имя функции.

Можно вызывать метод в контексте объекта следующим обра- зом:

object.methodname (params);

Создание Новых Объектов

И клиент и сервер JavaScript имеют строки предопределенных объектов. Кроме того, можно создавать собственные объекты. Соз- дание собственного объекта требует двух шагов:

1. Определить тип объекта, написанной функции.
2. Создать образец объекта с new.

Чтобы определять тип объекта, необходимо создать функцию для типа объекта, которая определяет его имя, и его свойства и ме- тоды. Например, пусть необходимо создавать тип объекта для авто- мобилей. Тип объектов будет назван car, и необходимо, чтобы он имел

свойства для make, model, year, и color. Чтобы сделать это, не- обходимо написать следующую функцию:

function car(make, model, year) { this.make = make;

this.model = model; this.year = year;

}

Замечание, используйте this, чтобы присвоить значения свойст- вам объекта, основанные на значениях функции.

Теперь можно создавать объект, с именем mycar следующим об- разом:

mycar = new car("Eagle", "Talon TSi", 1993);

Объект может иметь свойство, которое является самостоятель- ным другим объектом.

Можно определять методы для типа объекта включением опре- деление метода на определении типа объекта. Например, пусть есть набор файлов изображений GIF, и необходимо определить метод, который показывает информацию для car, наряду с соответствую-

щим изображением. Для этого необходимо определить функцию типа:

function displayCar() {

var result = "A Beautiful " + this.year

+ " " + this.make + " " +

this.model;

pretty_print(result)

}

Где pretty_print - предопределенная функция, которая показывает строку. Используйте this, чтобы обратиться к объекту, который принадлежит методу.

Далее необходимо определить функцию методом из car, прибав- ляя утверждение

This.displayCar = displayCar;

к определению объекта. Так, полное определение car теперь вы- глядит так:

function car(make, model, year, owner) { this.make = make;

this.model = model; this.year = year; this.owner = owner;

this.displayCar = displayCar;

}

Новый метод можно вызывать следующим образом:

car1.displayCar () car2.displayCar ()

Использование this для Ссылок Объекта

JavaScript имеет специальное ключевое слово, this, которое ис- пользуется, чтобы обращаться к текущему объекту. Например, есть функция с именем validate , которая проверяет правильность свой- ства значения объекта, данного объект, и high и low значения:

function validate(obj, lowval, hival) {

if ((obj.value < lowval) || (obj.value > hival)) alert("Invalid Value!")

Вызывать validate можно в каждом элементе формы обработчика событий onChange, используя this, как показано в следующем при- мере:

< INPUT TYPE = "text"

NAME = "age"

SIZE = 3

onChange="validate(this, 18, 99)">

Вообще, метод this обращается к вызывающему объекту.

ПК-1.2.3	Обучающийся владеет: инструментальными средствами, методами и навыками разработки программного обеспечения с использованием языка программирование высокого уровня.

СЕТЕВОЕ ПРОГРАММИРОВАНИЕ С СОКЕТАМИ И КАНАЛАМИ.

Цель работы: приобретение навыков проектирования и разработки приложений в архитектуре клиент-сервер.

Изучаемые вопросы

1. Тестирование программ без сети.
2. Порт
3. Сокеты
4. Простейший сервер и клиент

Постановка задачи

Создать на основе сокетов клиент/серверное приложение.

Теоретические сведения

Клиент-сервер (англ.Client-server) — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг (сервисов), называемыми серверами, и заказчиками услуг, называемыми клиентами. Нередко клиенты и серверы взаимодействуют через компьютерную сеть и могут быть как различными физическими устройствами, так и программным обеспечением.

Таким образом, работа сервера состоит в прослушивании соединения, она выполняется с помощью специального объекта, который вы создаете. Работа клиента состоит в попытке создать соединение с сервером, и это выполняется с помощью специального клиентского объекта, который вы создаете. Как только соединение установлено, вы увидите, что и клиентская, и серверная сторона соединения превращаются в потоковый объект ввода/вывода, таким образом вы можете трактовать соединение, как будто вы читаете и пишете файл. Таким образом, после установки соединения, вы просто используете хорошо знакомые команды ввода/вывода. Это одна из особенностей работы по сети в Java.

Порт

IP адреса не достаточно для уникальной идентификации сервера, так как на одной машине может существовать несколько серверов. Каждая IP машина также содержит порты, и когда вы

устанавливае- те клиент или сервер, вы должны выбрать порт, через который и клиент, и сервер согласны соединиться.

Порт - это программная абстракция. Клиентская программа зна- ет, как соединится с машиной через ее IP адрес, но она не может соединится с определенной службой на этой машине. Таким обра- зом номер порта стал вторым уровнем адресации. Идея состоит в том, что при запросе определенного порта вы запрашиваете службу, ассоциированную с этим номером порта. Обычно каждая служба ассоциируется с уникальным номером порта на определенной серверной машине. Клиент должен предварительно знать, на каком порту запущена нужная ему служба.

Системные службы зарезервировали использование портов с но- мерам от 1 до 1024, так что вы не можете использовать этот или любой другой порт, про который вы знаете, что он задействован.

Сокеты

Сокет - это программный интерфейс, предназначенный для пе- редачи данных между приложениями. Что же касается типов соке- тов, то их два - потоковые и датаграммные.

В Java вы создаете сокет, чтобы создать соединение с другой машиной, затем вы получаете InputStream и OutputStream (или, с соответствующими конверторами, Reader и Writer) из сокета, чтобы получить возможность трактовать соединение, как объект потока ввода/вывода. Существует два класса сокетов, основанных на пото- ках: ServerSocket, который использует сервер для "прослушивания" входящих соединений, и Socket, который использует клиент для инициализации соединения. Как только клиент создаст сокетное соединение, ServerSocket возвратит (посредством метода accept( )) соответствующий Socket, через который может происходить ком- муникация на стороне сервера. На этой стадии вы используете ме- тоды getInputStream( ) и getOutputStream( ) для получения соответ- ствующих объектов InputStream'a и outputStream'a для каждого со- кета. Они должны быть обернуты внутрь буферных и

форматирующих классов точно так же, как и другие объекты пото- ков.

Когда вы создаете ServerSocket, вы даете ему только номер пор- та. Вы не даете ему IP адрес, поскольку он уже есть на той машине, на которой он установлен. Однако когда вы создаете Socket, вы должны передать ему и IP адрес, и номер порта, с которым вы хоти- те соединиться. Socket, который возвращается из метода ServerSocket.accept( ) уже содержит всю эту информацию.

Простейший сервер и клиент

Этот пример покажет простейшее использование серверного и клиентского сокета. Все, что делает сервер, это ожидает соедине- ния, затем использует сокет, полученный при соединении, для соз- дания InputStream'a и OutputStream'a. Они конвертируются в Reader и Writer, которые оборачиваются в BufferedReader и PrintWriter. По- сле этого все, что будет прочитано из BufferedReader'a будет пере- правлено в PrintWriter, пока не будет получена строка "END", означающая, что пришло время закрыть соединение.

Клиент создает соединение с сервером, затем создает OutputStream и создает некоторую обертку, как и в сервере. Строки текста посылаются через полученный PrintWriter. Клиент также создает InputStream (опять таки, с соответствующей конвертацией и оберткой), чтобы слушать, что говорит сервер (который, в данном случае, просто отсылает слова назад).

Сервер, который просто отсылает назад все, что посылает кли- ент:

```
import java.io.*; import java.net.*;

public class ExampleServer {
```

```
// Выбираемпортвнепределов 1-1024: public static final int PORT = 8080;
public static void main(String[] args) throws IOException {
ServerSocket s = new ServerSocket(PORT); System.out.println("Started: " + s);
try { /* Блокирует до тех пор, пока не возник-
нетсоединение:*/
Socket socket = s.accept(); try {
System.out.println("Connection accepted:
" + socket);
BufferedReader in = new BufferedRead-
er(new InputStreamReader(
socket.getInputStream())); /* Выводавтоматическивыталкиваетсяизбуфера PrintWriter'ом*/
PrintWriter out = new PrintWriter(new BufferedWriter(
new OutputStreamWri- ter(socket.getOutputStream())), true);
while (true) {
String str = in.readLine(); if (str.equals("END"))
break;
System.out.println("Echoing: " + str); out.println(str);
}
/* Всегда закрываем два сокета...*/
}
finally {
System.out.println("closing..."); socket.close();
}
}
finally {
s.close();
}
}
}
```

Вы можете видеть, что для ServerSocket'а необходим только но- мер порта, а не IP адрес (так как он запускается на локальной ма- шине!). Когда вы вызываете accept( ), метод блокирует выполнение до тех пор, пока клиент не попробует подсоединится к серверу. То есть, сервер ожидает соединения, но другой процесс может выпол- няться. Когда соединение установлено, метод accept( ) возвращает объект Socket, представляющий это соединение.

Здесь тщательно обработана ответственность за очистку сокета. Если конструктор ServerSocket завершится неудачей, программа просто звершится (обратите внимание, что мы должны

предполо- жить, что конструктор ServerSocket не оставляет никаких открытых сокетов, если он зваершается неудачей). По этой причине main( ) выбрасывает IOException, так что в блоке try нет необходимости. Если конструктор ServerSocket завершится успешно, то все вызовы методов должны быть помещены в блок try-finally, чтобы убедить- ся, что блок не будет покинут ни при каких условиях и ServerSocket будет правильно закрыт.

Аналогичная логика используется для сокета, возвращаемого из метода accept( ). Если метод accept( ) завершится неудачей, то мы должны предположить, что сокет не существует и не удерживает никаких ресурсов, так что он не нуждается в очистке. Однако если он закончится успешно, то следующие выражения должны быть помещены в блок try-finally, чтобы при каких-либо ошибках все равно произошла очистка. Позаботится об этом необходимо, пото- му что сокеты используют важные ресурсы, не относящиеся к памя- ти, так что вы должны быть прилежны и очищать их (так как в Java нет деструкторов, чтобы сделать это за вас).

И ServerSocket и Socket, производимый методом accept( ), печа- таются в System.out. Это означает, что автоматически вызывается их метод toString( ). Вотчтоонвыдаст:
ServerSocket[addr=0.0.0.0,PORT=0,localport=8080]
Socket[addr=127.0.0.1,PORT=1077,localport=8080]

Следующая часть программы выглядит, как открытие файла для чтения и записи за исключением того, что InputStream и OutputStream создаются из объекта Socket. И объект InputStream'a и OutputStream'a конвертируются в объекты Reader'a и Writer'a с по- мощью "классов-конвертеров" InputStreamReader и OutputStreamreader, соответственно. Вы можете также использовать классы InputStream и OutoutStream напрямую, но, с точки зрения вывода, есть явное преимущество в использовании этого подхода. Оно проявляется в PrintWriter'e, который имеет перегруженный конструктор, принимающий в качестве второго аргумента флаг ти- па boolean, указывающий, нужно ли автоматическое выталкивание буфера вывода в конце каждого выражения println( ) (но не print( )). Каждый раз, когда вы записываете в вывод, буфер вывода должен

выталкиваться, чтобы информация проходила по сети. Выталкива- ние важно для этого конкретного примера, поскольку клиент и сер- вер ожидают строку от другой стороны, прежде, чем приступят к ее обработке. Если выталкивание буфера не произойдет, информация не будет помещена в сеть до тех пор, пока буфер не заполнится, что может привести к многочисленным проблемам в этом примере.

Когда пишите сетевую программу, вам необходимо быть осто- рожным при использовании автоматического выталкивания буфера. При каждом выталкивании буфера пакеты должны создаваться и отправляться. В данном случае это именно то, что нам надо, так как если пакет, содержащий строку, не будет отослан, то общение меж- ду сервером и клиентом остановится. Другими словами, конец строки является концом сообщения. Но во многих случаях, сообще- ния не ограничиваются строками, так что будет более эффективным использовать автоматическое выталкивание буфера, поэтому по- звольте встроенному механизму буфферизации построить и ото- слать пакет. В таком случае могут быть посланы пакеты большего размера и процесс обработки пойдет быстрее.

Обратите внимание, что фактически все открытые вами потоки, буфферезированы.

В бесконечном цикле while происходит чтение строк из входного BufferedReader'a и запись информации в System.out и в выходной PrintWriter. Обратите внимание, что вход и выход могут быть лю- быми потоками, так случилось, что они связаны с сетью.

Когда клиент посылает строку, содержащую "END", про- грамма прекращает цикл и закрывает сокет.

Клиент, который просто посылает строки на сервер и чита- ет строки, посылаемые сервером:

```
import java.net.*; import java.io.*;
public class ExampleClient {
public static void main(String[] args) throws IOException {
// Передаем null в getByName(), получая
// специальный IP адрес "локальной заглушки"
// для тестирования на машине без сети:
InetAddress addr = InetAddress.getByName(null);
// Альтернативно, вы можете использовать
// адрес или имя:
// InetAddress addr =
// InetAddress.getByName("127.0.0.1");
// InetAddress addr =
// InetAddress.getByName("localhost"); System.out.println("addr = " + addr);
Socket socket = new Socket(addr, JabberServ- er.PORT);
// Помещаем все в блок try-finally, чтобы
// быть уверенным, что сокет закроется: try {
System.out.println("socket = " + socket); BufferedReader in = new BufferedReader(new
InputStreamReader(socket
.getInputStream()));
// Выводавтоматическивыталкивается
PrintWriter'ом.
PrintWriter out = new PrintWriter(new Buffe- redWriter(
new OutputStreamWri- ter(socket.getOutputStream())), true);
for (int i = 0; i < 10; i++) { out.println("howdy " + i); String str = in.readLine();
System.out.println(str);
}
out.println("END");
}
finally {
System.out.println("closing..."); socket.close();
}
}
}
```

В main( ) вы можете видеть все три способа получения IP адреса локальной заглушки: с помощью null, localhost или путем явного указания зарезервированного адреса 127.0.0.1, если

вы хотите со- единится с машиной по сети, вы замените его IP адресом машины.

Обратите внимание, что Socket создается при указании и InetAddress'a, и номера порта. Чтобы понять, что это значит, когда будете писать один из объектов Socket помните, что Интернет со-единение уникально определяется четырьмя параметрами: клиент- ским хостом, клиентским номером порта, серверным хостом и сер- верным номером порта. Когда запускается сервер, он получает на- значаемый порт (8080) на localhost (127.0.0.1). Когда запускается клиент, он располагается на следующем доступном порту на своей машине, 1077 - в данном случае порт, который так же оказался на той же самой машине (127.0.0.1), что и сервер. Теперь, чтобы пере-дать данные между клиентом и сервером, каждая сторона знает, ку- да посылать их. Поэтому, в процессе соединения с "известным" сервером клиент посылает "обратный адрес", чтобы сервер знал, куда посылать данные. Вот что вы видите среди выводимого сторо- ной сервера:

Socket[addr=127.0.0.1,port=1077,localport=8080]

Это означает, что сервер просто принимает соединение с адреса

127.0.0.1 и порта 1077 во время прослушивания локального порта

(8080). Наклиентскойстороне:

Socket[addr=localhost/127.0.0.1,PORT=8080,localport

=1077]

Это значит, что клиент установил соединение с адресом 127.0.0.1

по порту 8080, используя локальный порт 1077.

Вы заметите, что при каждом повторном запуске клиента номер локального порта увеличивается. Он начинается с 1025 (первый по- сле зарезервированного блока портов) и будет увеличиваться до тех пор, пока вы не перезапустите машину, в таком случае он снова начнется с 1025. (На машинах под управлением UNIX, как только будет достигнут верхний предел диапазона сокетов, номер будет возвращен снова к наименьшему доступному номеру.)

Как только объект Socket будет создан, процесс перейдет к BufferedReader и PrintWriter, как мы это уже видели в сервере (опять таки, в обоих случаях вы начинаете с Socket'a). В данном случае, клиент инициирует обмен путем посылки строки "howdy", за кото- рой следует число. Обратите внимание, что буфер должен опять выталкиваться (что происходит автоматически из-за второго аргу- мента в конструкторе PrintWriter'a). Если буфер не будет выталки- ваться, процесс обмена повиснет, поскольку начальное "howdy" ни-

когда не будет послана (буфер недостаточно заполнен, чтобы от- сылка произошла автоматически). Каждая строка, посылаемая назад сервером, записывается в System.out, чтобы проверить, что все ра- ботает корректно. Для завершения обмена посылается ранее огово-ренный "END". Если клиент просто разорвет соединение, то сервер выбросит исключение.

Вы можете видеть, что аналогичные меры приняты, чтобы быть уверенным в том, что сетевые ресурсы, представляемые сокетом, будут правильно очищены. Для этого используется блок try-finally.

Задание

Создать на основе сокетов клиент-серверное приложение:

1. Клиент посылает через сервер сообщение другому клиен- ту.
2. Клиент посылает через сервер сообщение другому клиен- ту, выбранному из списка.
3. Чат. Клиент посылает через сервер сообщение, которое по- лучают все клиенты. Список

клиентов хранится на сервере в файле.

4. Клиент при обращении к серверу получает случайно вы- бранный сонет Шекспира из файла.

5. Сервер рассылает сообщения выбранным из списка кли- ентам. Список хранится в файле.

6. Сервер рассылает сообщения в определенное время опре- деленным клиентам.

7. Сервер рассылает сообщения только тем клиентам, кото- рые в настоящий момент находятся в on-line.

8. Чат. Сервер рассылает всем клиентам информацию о кли- ентах, вошедших в чат и покинувших его.

9. Клиент выбирает изображение из списка и пересылает его другому клиенту через сервер.

10. Игра по сети в “Морской бой”.

11. Игра по сети “Го”. Крестики-нолики на безразмерном (большом) поле. Для победы необходимо выстроить пять в один ряд.

12. Написать программу, сканирующую сеть в указанном диапазоне IP адресов на наличие активных компьютеров.

13. Прокси. Программа должна принимать сообщения от лю- бого клиента, работающего на протоколе TCP, и отсылать их на соответствующий сервер. При передаче изменять сообщение.

14. Телнет. Создать программу, которая соединяется с ука- занным сервером по указанному порту и производит об- мен текстовой информацией.

3. Методические материалы, определяющие процедуру и критерии оценивания сформированности компетенций при проведении промежуточной аттестации

Критерии формирования оценок по ответам на вопросы, выполнению тестовых заданий

- оценка **«отлично»** выставляется обучающемуся, если количество правильных ответов на вопросы составляет 100 – 90% от общего объёма заданных вопросов;

- оценка **«хорошо»** выставляется обучающемуся, если количество правильных ответов на вопросы – 89 – 76% от общего объёма заданных вопросов;

- оценка **«удовлетворительно»** выставляется обучающемуся, если количество правильных ответов на тестовые вопросы –75–60 % от общего объёма заданных вопросов;

- оценка **«неудовлетворительно»** выставляется обучающемуся, если количество правильных ответов – менее 60% от общего объёма заданных вопросов.

Критерии формирования оценок по результатам выполнения заданий

«Отлично/зачтено» – ставится за работу, выполненную полностью без ошибок и недочетов.

«Хорошо/зачтено» – ставится за работу, выполненную полностью, но при наличии в ней не более одной негрубой ошибки и одного недочета, не более трех недочетов.

«Удовлетворительно/зачтено» – ставится за работу, если обучающийся правильно выполнил не менее 2/3 всей работы или допустил не более одной грубой ошибки и двух недочетов, не более одной грубой и одной негрубой ошибки, не более трех негрубых ошибок, одной негрубой ошибки и двух недочетов.

«Неудовлетворительно/не зачтено» – ставится за работу, если число ошибок и недочетов превысило норму для оценки «удовлетворительно» или правильно выполнено менее 2/3 всей работы.

Виды ошибок:

- грубые ошибки: незнание основных понятий, правил, норм; незнание приемов решения задач; ошибки, показывающие неправильное понимание условия предложенного задания.

- негрубые ошибки: неточности формулировок, определений; нерациональный выбор хода решения.

- недочеты: нерациональные приемы выполнения задания; отдельные погрешности в формулировке выводов; небрежное выполнение задания.

Критерии формирования оценок по зачету с оценкой

«Отлично/зачтено» – студент приобрел необходимые умения и навыки, продемонстрировал навык практического применения полученных знаний, не допустил логических и фактических ошибок

«Хорошо/зачтено» – студент приобрел необходимые умения и навыки, продемонстрировал навык практического применения полученных знаний; допустил незначительные ошибки и неточности.

«Удовлетворительно/зачтено» – студент допустил существенные ошибки.

«Неудовлетворительно/не зачтено» – студент демонстрирует фрагментарные знания изучаемого курса; отсутствуют необходимые умения и навыки, допущены грубые ошибки.

Критерии формирования оценок по экзамену

«**Отлично**» (5 баллов) – обучающийся демонстрирует знание всех разделов изучаемой дисциплины: содержание базовых понятий и фундаментальных проблем; умение излагать программный материал с демонстрацией конкретных примеров. Свободное владение материалом должно характеризоваться логической ясностью и четким видением путей

применения полученных знаний в практической деятельности, умением связать материал с другими отраслями знания.

«**Хорошо**» (4 балла) – обучающийся демонстрирует знания всех разделов изучаемой дисциплины: содержание базовых понятий и фундаментальных проблем; приобрел необходимые умения и навыки, освоил вопросы практического применения полученных знаний, не допустил фактических ошибок при ответе, достаточно последовательно и логично излагает теоретический материал, допуская лишь незначительные нарушения последовательности изложения и некоторые неточности. Таким образом данная оценка выставляется за правильный, но недостаточно полный ответ.

«**Удовлетворительно**» (3 балла) – обучающийся демонстрирует знание основных разделов программы изучаемого курса: его базовых понятий и фундаментальных проблем. Однако знание основных проблем курса не подкрепляются конкретными практическими примерами, не полностью раскрыта сущность вопросов, ответ недостаточно логичен и не всегда последователен, допущены ошибки и неточности.

«**Неудовлетворительно**» (0 баллов) – выставляется в том случае, когда обучающийся демонстрирует фрагментарные знания основных разделов программы изучаемого курса: его базовых понятий и фундаментальных проблем. У экзаменуемого слабо выражена способность к самостоятельному аналитическому мышлению, имеются затруднения в изложении материала, отсутствуют необходимые умения и навыки, допущены грубые ошибки и незнание терминологии, отказ отвечать на дополнительные вопросы, знание которых необходимо для получения положительной оценки.

Экспертный лист
оценочных материалов для проведения промежуточной аттестации по
дисциплине «Программирование сетевых задач»

по направлению подготовки/специальности

09.03.03 Прикладная информатика
(код и наименование)

Направленность (профиль)/специализация

Прикладная информатика на железнодорожном транспорте
(наименование)

Бакалавр
квалификация выпускника

1. Формальное оценивание			
Показатели		Присутствуют	Отсутствуют
Наличие обязательных структурных элементов:		+	
–титульный лист		+	
–пояснительная записка		+	
– типовые оценочные материалы		+	
–методические материалы, определяющие процедуру и критерии оценивания		+	
Содержательное оценивание			
Показатели	Соответствует	Соответствует частично	Не соответствует
Соответствие требованиям ФГОС ВО к результатам освоения программы	+		
Соответствие требованиям ОПОП ВО к результатам освоения программы	+		
Ориентация на требования к трудовым функциям ПС (при наличии утвержденного ПС)	+		
Соответствует формируемым компетенциям, индикаторам достижения компетенций	+		

Заключение: ФОС рекомендуется/ не рекомендуется к внедрению; обеспечивает/ не обеспечивает объективность и достоверность результатов при проведении оценивания результатов обучения; критерии и показатели оценивания компетенций, шкалы оценивания обеспечивают/ не обеспечивают проведение всесторонней оценки результатов обучения.

Эксперт, должность, ученая степень, ученое звание _____ /

(подпись)