

Веб-технологии

Захар Кириллов, MSc

Tarkvaraarenduse lektor
zahhar.kirillov@mk.ee

Учебный план 1/3

- 24 академических часа лекций и практикумов
- 48 часов самостоятельной работы
 - Домашнее задание (*объёмное!*)
 - Работа с документацией и дополнительными учебными материалами
 - Википедия и мануалы :)
- Экзамен
 - Практическое задание
 - тест

Учебный план 2/3

- 6 очных встреч по 4 акад. часа:
 - 5 февраля в 11:00
 - 11 февраля в 17:30
 - 28 февраля в 17:30
 - 7 марта в 17:30
 - 18 марта в 17:30
 - 24 марта в 17:30
- домашние работы (от 2 до 4)
 - Задания публикуются на mainor.info
 - Ответы принимаются через LePress там же
 - Потребуется доступ на FTP (porthos.mk.ee)

Учебный план 3/3

Экзамен

- Допускаются только те, кто вовремя сдаст выполненные домашние работы
- Состоится в начале марта (*нужно договориться о датах!*)
- 40% результата обеспечивает тест по основным теоретическим вопросам курса (*20 вопросов за 30 минут*);
- Остальные 60% даёт практическое задание (*45 минут, сразу после теста*)
- Активная работа на лекциях может дать +1 балл на экзамене

Каков ваш уровень?

- Что это такое? С чем из перечисленного вам приходилось сталкиваться? С чем вы разобрались самостоятельно до уровня хорошего понимания и использования?
 - IE/Mozilla/Chrome/Opera/Safari
 - HTML/CSS/XML
 - FTP/DNS/HTTP
 - Apache, nginx, IIS
 - Perl, PHP, Python, RoR, .NET, J2EE
 - Javascript, Ajax

Что мы будем изучать?

- Веб (WWW, всемирная паутина, Сеть) - *“распределенная система, предоставляющая доступ к связанным между собой документам, расположенным на различных компьютерах, подключенных к Интернету” (W) = 1,5-2 млрд пользователей*
- Технология (от греч. “τέχνη” — искусство, мастерство, умение; + “λόγος” — мысль, причина”) - *комплекс организационных мер, операций и приемов, направленных на изготовление, обслуживание, ремонт и/или эксплуатацию изделия с номинальным качеством и оптимальными затратами (W)*

Что такое веб-технологии?

Веб-технологии (или веб-технология) – это комплекс технических, коммуникационных и программных методов для организации совместной деятельности пользователей в интернете.

В чем суть “веб-технологий”?

- Главное – понять концепции и принципы работы в интернете “за окном браузера”:
 - Создание информации пользователями
 - Размещение её в сети и предоставление доступа к ней
 - Контролируемое распространение
 - Взаимодействие с другими информационными ресурсами интернета
 - Понимание технических процессов, которые за всем этим стоят

Для чего мы это изучаем?

- “Информация правит миром”
- Веб – на сегодняшний день наиболее универсальная и благоприятная среда распространения информации (*оперативность, отсутствие границ, мультязычность, -культурность, -медийность, дешевизна и т.п.*)
- Использование веба в качестве интерфейса между человеком и компьютером постоянно расширяется (*cloud computing, загружаемые из интернета опсистемы, iPhone/iPad...*)

Разберемся “по понятиям” 1/3

- Веб, интернет, Сеть, WWW = синонимы
- Контент = информация, размещенная в Сети
- Браузер – компьютерная программа, при помощи которой “обычные пользователи” выходят в сеть (MSIE, FF, Chrome, Opera, ...)
- Страница (гипертекстовый документ) – то, что показывает браузер после ввода адреса или клика на гиперссылку (например: <https://moodle.mk.ee/course/view.php?id=86>)
- Сайт - совокупность связанных гиперссылками страниц, представляющих собой единое целое в техническом, информационном и навигационном аспектах и объединенных общим уникальным именем (например, www.mk.ee)

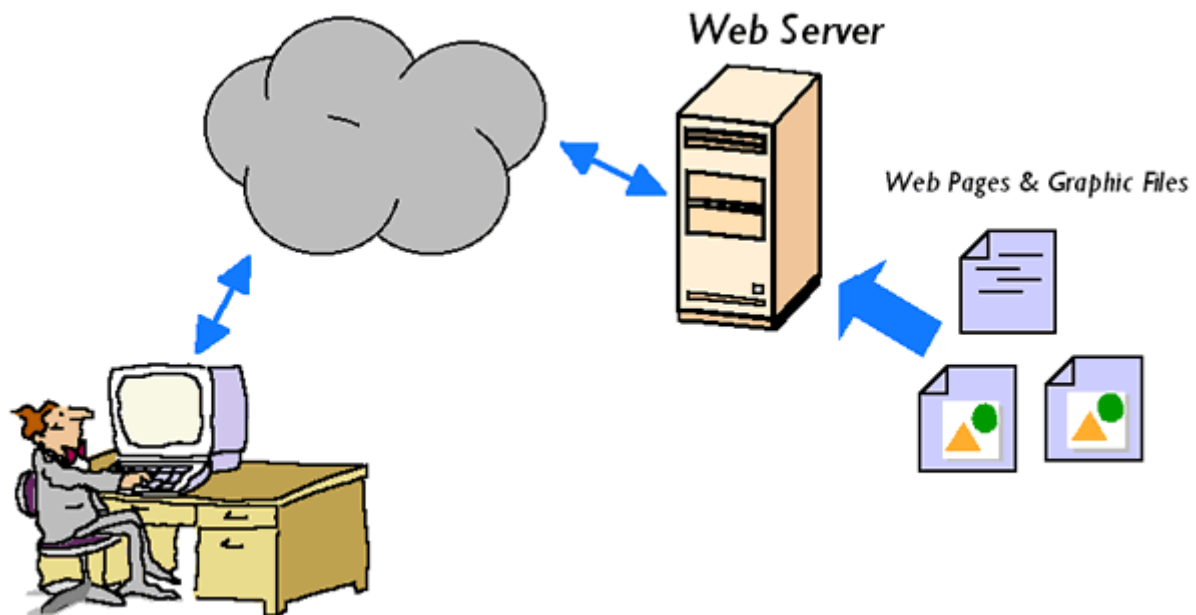
Разберемся “по понятиям” 2/3

- Портал – крупный сайт, предоставляющий своим посетителям ряд сервисов или способов взаимодействия и являющийся для многих “точкой отправления” в интернет (например: google.com, yandex.ru)
- Сервис (служба) – размещенная на сайте компьютерная программа для решения специфической задачи, имеющая стандартизированный интерфейс для взаимодействия с человеком или другими программами (например, авторизация).

Разберемся “по понятиям” 3/3

- Сервер – идентифицируемый “узел” интернета; аппаратная часть представляет собой подключенный к интернету компьютер (как правило, не являющийся рабочим местом какого-то определенного пользователя и с чуть более специфическим железом по сравнению с обычными рабочими станциями), а программная – комплекс ПО для параллельной обработки множества поступающих по определенным интернет-протоколам запросов

Как работает интернет?



Архитектура “клиент-сервер”

- Интернет работает не постоянным непрерывным потоком, а по системе “запрос-ответ”
- Одна сторона (**клиент, front-end**) инициирует запрос (устанавливает связь), другая (**сервер, back-end**) отвечает на запрос, возвращая запрошенные данные или своё состояние и разрывает сеанс связи (или он будет разорван клиентом по истечении времени ожидания ответа, т.н. **time-out**).
- “Клиент” и “сервер” - понятия относительные (один сервер может обращаться к другому, становясь его клиентом)

“Толстый” и “тонкий” клиенты

- Чем более “сырые” данные отдаёт клиенту сервер, тем больше работы по обработке и представлению этих данных потребителю должен взять на себя клиент. В этом случае говорят, что **клиент толстый**;
 - И наоборот:
- Чем больше работы по подготовке данных проводится на стороне сервера (*например, упорядочивание списка по алфавиту перед выдачей*), тем меньше “интеллекта” (а также времени и ресурсов) требуется от клиента и потому он называется **тонким клиентом**

Браузер = клиент для веба

- Работает под достаточно жёстким контролем операционной системы (в целях безопасности пользовательских данных ограничен доступ к файловой системе, устройствам ввода-вывода и т.п.)
- Распространяется бесплатно (но не обязательно с открытым исходным кодом!)
- Является причиной многих *holy wars* и головных болей веб-разработчиков, т.к. могут немного по-разному отображать одну и ту же страницу (код)
- Нуждается в постоянном обновлении (*кстати да – проверьте свою версию!*)

Вопрос:

Браузер – это *тонкий* или *толстый* клиент?

По мере развития веба компьютер в целом становится всё более тонким клиентом (*cloud computing*), а браузер – более толстым клиентом (*Ajax, Javascript* – позволяют серверу выдавать “голые” данные, подготовку к представлению которых полностью берет на себя клиент)

Это позволяет эффективно распределить функциональность (пользователь некомпетентен; рабочая станция - ненадёжна) и нагрузку в сети (к серверу обращаются тысячи пользователей, каждый из которых обладает внушительной вычислительной мощностью). Каждая сторона берёт на себя ту часть, в которой она наиболее сильна.

Современные браузеры

- Графические браузеры:
 - Microsoft Internet Explorer (только в среде Windows), текущая версия - 8.0;
 - Mozilla Firefox (свободное ПО), версия 3.6.x
 - Google Chrome (свободное ПО), версия 9.x;
 - Safari (Mac/Win), версия 5.x;
 - Opera, версия 11.x
 - + целый ряд малораспространенных имён
- Текстовые браузеры (для терминалов)
- Мобильные браузеры (для iPhone, Symbian, Android и других устройств)

Статистика браузеров

- В Эстонии > 90% это Firefox+Chrome
- В Южной Корее > 90% это MS IE
- В мире:
http://www.w3schools.com/browsers/browsers_stats.asp
 - Доля Chrome и Safari растёт
 - Доля Firefox стабильна (будет уменьшатся?)
 - Доля IE уменьшается, но присутствует три “поколения”: 6.0 (*проблема!*), 7.0 и 8.0
 - Opera имеет свой фан-клуб (страны СНГ)
- А какой браузер используете вы? Почему?

Как браузер отображает веб-страницу?

- На самом деле вместо браузера веб-старницу рисует на экране т.н. “движок” или *layout engine*
- Всего существует 4 различных движка:
 - Trident от Microsoft (коммерческий)
 - Gecko от Mozilla (свободный)
 - WebKit от Apple/Nokia/Google (свобоный)
 - Presto от Opera (коммерческий)
- Используются везде, где нужно отобразить (*рендерить*) веб-страницы: почтовые клиенты, браузеры игровых консолей и смартфонов, HTML-редакторы и другой прикладной софт.

Как клиент (браузер) общается с сервером?

- В основе общения – протокол обмена данными
- Протокол - набор соглашений и стандартов, определяющий порядок и способы передачи сообщений и обработки ошибок между различными программами (или “железом”)
- Веб работает с протоколами уровня данных (*те, что после TCP/IP* :) по модели OSI:
 - Session (PPTP, RTSP, L2TP, H.245 и др.)
 - Presentation (ASCII)
 - Application (HTTP, FTP, SMTP)

Идеальный протокол

- Может быть реализован на разных ЯП/железе
- Либеральный при приёме сообщений (“прощает” неточности передающей стороны)
- Строгий при передаче сообщений (отправляет исчерпывающую информацию)
- Информировывает другую сторону о состоянии сеанса связи
- Может обнаружить ошибку передачи и попытаться её исправить (самостоятельно или запросив повторную передачу)
- Не допускает несанкционированного вмешательства третьей стороны
- Не существует :)

Протокол HTTP

- HTTP = Hypertext Transport Protocol
- Протокол прикладного уровня (OSI application layer 7)
- Основной протокол обмена данными в интернете (по объему трафика)
- Архитектурно суть “клиент-сервер” (запрос/ответ)
- Может быть транспортом для других протоколов (потокое аудио/видео, XML-RPC, SOAP, e-Donkey и др)
- *Stateless* – не сохраняет состояния между запросами и ответами от одного клиента (следующий запрос ничего не знает о предыдущем) – это отдано на откуп прикладному ПО (браузеру и веб-серверу)

История и версии HTTP

- Создан Тимом Бёрнерсом-Ли в 1990-92 гг
- Версия 1.0 (IETF RFC1945) в мае 1996 г
- Версия 1.1 в июне 1999 г (нынешняя):
 - Имя хоста стало обязательным (позволило разместить несколько сайтов на одном компьютере – виртуальный хостинг)
 - Появилась возможность “удерживать” соединение с сервером, посылая несколько запросов за один сеанс связи (*но кол-во доступных соединений в единицу времени ограничено, что может стать проблемой!*)

Суть протокола HTTP

- Клиент запрашивает у сервера определенный *документ* по его известному уникальному идентификатору (URI)
- Сервер отвечает клиенту либо выдавая запрашиваемый документ, либо информируя о состоянии документа (“не найден”, “перемещён”, “нет прав для доступа” и т.д.)
- Протокол очень прост в реализации – полное описание последней версии занимает всего 60 страниц.

URI

- URI = Unified (Universal) Resource Identifier
- Задумывался как универсальный идентификатор всего-чего-угодно в виртуальном и реальном мире (*человек, дом, книга, файл, сайт и т.п.*)
- Суть символьная строка, синтаксис:
 - <схема>:<идентификатор-по-схеме>
 - *Схема* определяет тип ресурса
 - *Идентификатор* может нести информацию об описываемом ресурсе или просто быть его порядковым номером (*ключом*)

Примеры URI

- ISBN:978-0-9613921-4-7 (*книги, включает информацию о языке и издательстве*)
- EAN:4741162002643 (*штрих-коды товаров, несет информацию о стране- и фирме-производителе товара*)
- mailto:zahhar@gmail.com
- skype:zahharkirillov?chat
- http://www.mk.ee
- ftp://pothos.mk.ee/~zahharkirillov

URL

- URL = Unified (Universal) Resource Locator
- URL = URI для интернет-ресурсов, тоже придуман Т.Бёрнерсом-Ли
- Суть символьная строка, синтаксис:

<схема>://<логин>:<пароль>@<хост>:<порт>/<путь>

- Схема = протокол (http, ftp, ...)
- // = избыточный бесполезный разделитель :)
- Необязательные логин:пароль@ для доступа к ресурсу
- Хост:порт = имя хоста или его IP-адрес и порт
- Путь к документу относительно корня хоста и дополнительные параметры: переменные, ссылка внутрь документа и т.п.

Примеры URL

- <http://mk.ee>
- <http://www.google.com/?q=mainor>
- ftp://username:passwod@examplecom:21/public_html/folder/filename.ext
- http://ru.wikipedia.org/wiki/URL#.D0.9F.D1.80.D0.B8.D0.BC.D0.B5.D1.80.D1.8B_URL

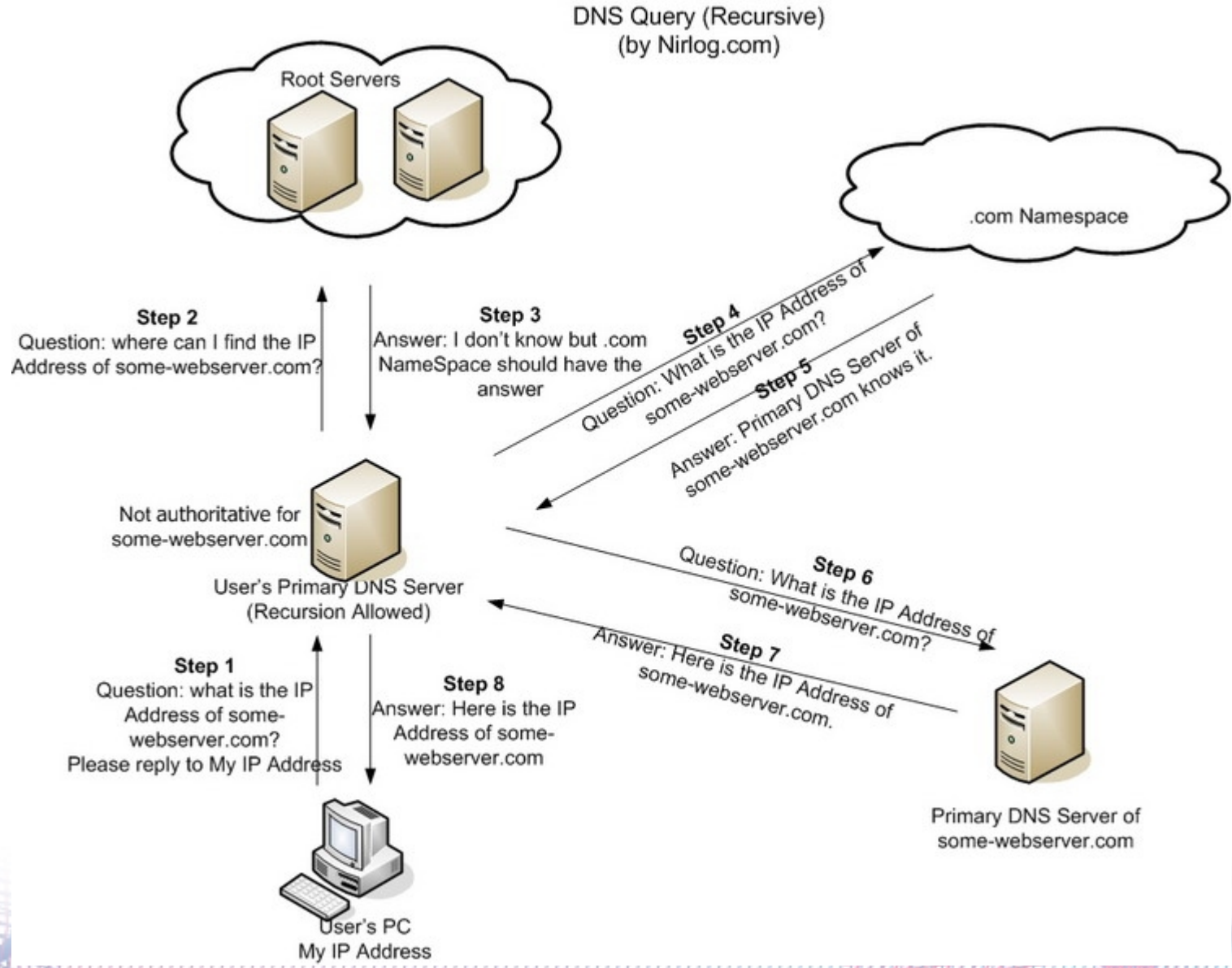
Проблемы URL

- Как правило длинный и плохо запоминающийся;
- Синтаксис с трудом понятен обычному человеку;
- Не обеспечивают уникальности: под одним и тем же URL в разное время могут скрываться разные документы (например, результаты поисковиков)
- Поддерживает ограниченный набор символов (недавно добавлена поддержка ряда нелатинских символов для http)

DNS

- Имя хоста в URL (и, соответственно, в HTTP-протоколе) может быть задано либо IP-адресом сервера, либо его доменным именем (*domain name, host name*).
- Соответствия между именами хостов (буквенные, легче запомнить) и IP-адресами (числовые) хранятся в распределенной системе DNS – Domain Name Service.
- Одному хосту соответствует один IP-адрес; но за одним IP-адресом фактически может находиться множество хостов.

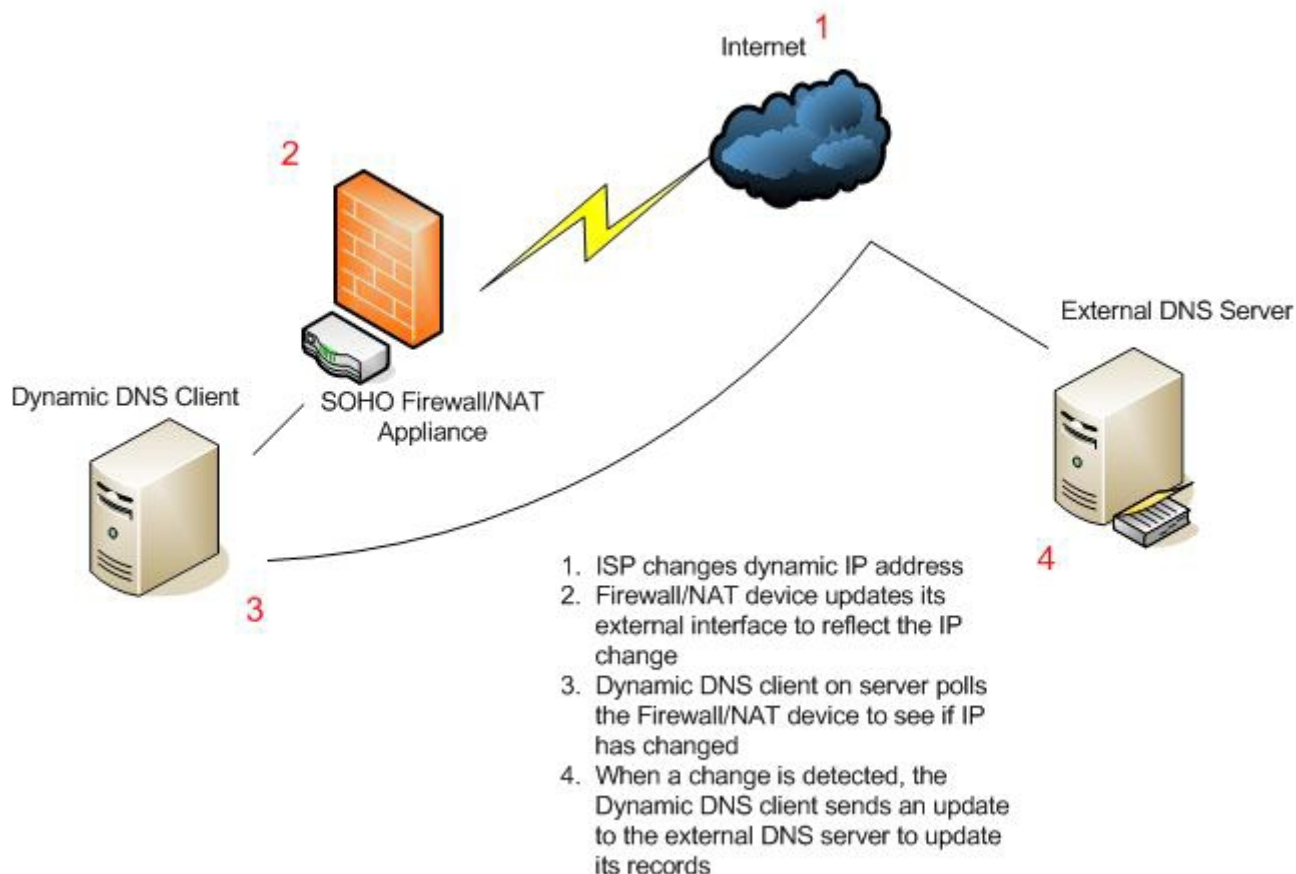
О чудо – вторая картинка! (показывает, как работает DNS)



Ещё кое-что о DNS

- Обычно мы используем DNS-сервис, предоставляемый нашим интернет-провайдером. Но мы можем воспользоваться DNS других провайдеров и публичными сервисами:
 - OpenDNS (208.67.222.222 / 208.67.220.220)
 - Google Public DNS (8.8.8.8 / 8.8.4.4)
- Статические и динамические (чтобы домашние пользователи тоже могли разместить у себя веб-сервер)
- Авторитетные (выдающие информацию по своей зоне) и кеширующие (временно хранящие статические адреса других востребованных хостов и зон)

Динамический DNS



- Динамические DNS поддерживает софт многих рутеров (DD-Wrt / OpenWRT)
- Предоставляется сервисами no-ip.com, dyndns.com, changeip.com и др

Утилиты для работы с DNS (и не только)

- Ping (тоже покажет IP по имени хоста)
- Tracert (покажет IP и путь пакетов к нему)
- Nslookup <host || ip> <ns>
 - Ns = адрес или хост DNS-сервера, на котором ищем; по-умолчанию общий DNS.
- Whois
 - (по-умолчанию не встроен в Windows, можно скачать здесь:
<http://technet.microsoft.com/en-us/sysinternals/bb897435.aspx>)

Вернемся к нашим баранам...

- Принцип работы HTTP “глазами” клиента:
 - Клиент устанавливает соединение с портом 80 веб-сервера по протоколу TCP
 - запрашивает документ по его URL
 - читает ответ сервера: состояние (ОК или ошибка) и содержимое документа
- “Глазами” сервера:
 - Сервер “слушает” запросы на определенном порту (это позволяет одновременно запускать на одном компьютере несколько различных копий серверного ПО)
 - Проверяет наличие прав доступа к документу и собственно документа
 - Отправляет клиенту ответ: код+документ (или сообщение об ошибке)

Пример запроса

```
GET /index.html HTTP/1.1  
Host: www.mk.com
```

Пример ответа

```
HTTP/1.1 200 OK  
Date: Fri, 04 Feb 2011 21:09:23 GMT  
Server: Apache/2.2.3 (Red Hat)  
X-Powered-By: PHP/5.1.6  
Set-Cookie:  
mk4=752ef2svl20pkmit7j8bu3vk66; path=/  
Expires: Thu, 19 Nov 1981 08:52:00 GMT  
Cache-Control: no-store, no-cache  
Pragma: no-cache  
Connection: close  
Transfer-Encoding: chunked  
Content-Type: text/html; charset=utf-8
```

Структура протокола HTTP

- Каждое сообщение состоит из 3 частей:
 - Starting line = тип сообщения (запрос или ответ)
 - Headers = параметры передачи данных
 - (пустая строка)
 - Message Body = тело сообщения
- Обязательна только *starting line* (и пустая строка, если передаётся *message body*)
- Порядок частей важен!

Стартовая строка запроса

Синтаксис запроса:

`<метод> <URL> HTTP/<версия>
<хост или IP-адрес>`

Например:

```
GET /index.php HTTP/1.1
```

```
Host: www.mk.ee
```

*попробуйте выполнить в telnet'e –
cmd → telnet mk.ee 80)*

Доступные методы: GET, HEAD
(обязательные), OPTIONS, TRACE
(безопасные); PUT, POST, DELETE.

Ответ сервера

HTTP/1.1 200 OK

Date: Mon, 23 May 2005 22:38:34 GMT

Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)

Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT

Content-Length: 438

Connection: close

Content-Type: text/html; charset=UTF-8

(пустая строка)

<содержимое запрошенного ресурса>

Стартовая строка ответа

- Синтаксис ответа

HTTP/<версия> <статус> <пояснение>

– Например:

HTTP/1.1 200 ОК

- Статусы

- 1xx запрос в обработке (предотвращает тайм-аут);
- 2xx запрос принят, понят и успешно обработан;
- 3xx для выполнения запроса нужно внимание клиента (перенаправление и т.п.)
- 4xx ошибка на стороне клиента
- 5xx ошибка на стороне сервера

Изучаем ответы серверов

- <http://www.serverheader.com/> через веб
- “Header Spy” || “liveHTTPheaders” Firefox estensions
- <http://www.httpwatch.com> плагин для MS IE
- Ditto add-on для Chrome

Что происходит на сервере?

- HTTP-запросы обрабатываются специальным ПО, называемым “веб-сервером” (*NB! Один термин и для софта, и для железа*)
- Наиболее популярные веб-серверы
 - Apache2 мультиплатформенный (~55%)
 - IIS для MS Windows Server (~25%)
 - Nginx (~8%), Lighttpd, Zope и др. свободные
 - Google Web Server (7%)
 - Проприетарные от IBM, Oracle, Apple, Adobe

“За кулисами веб-сервера”

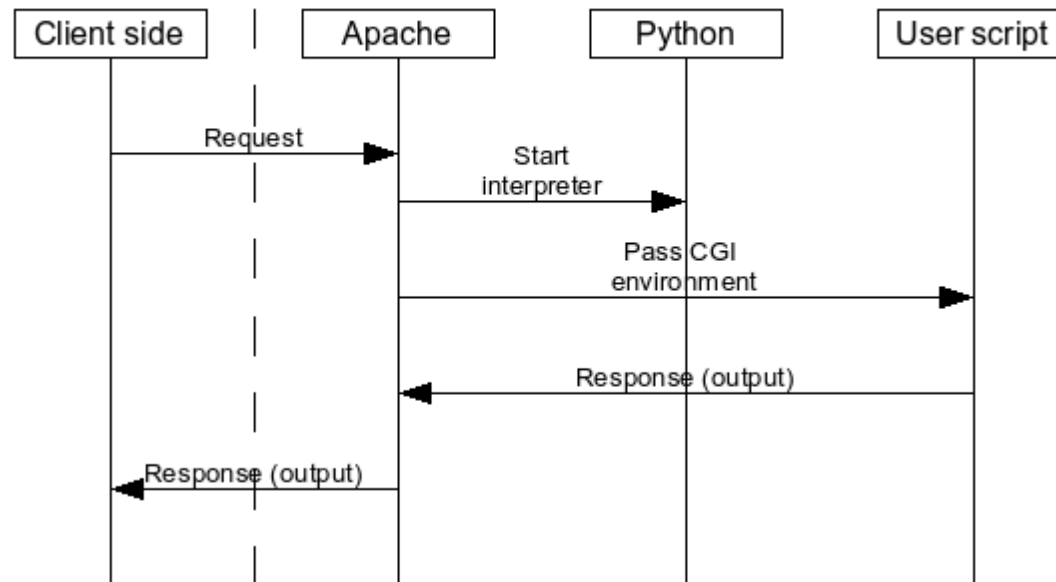
- Сам веб-сервер:
 - отвечает на запросы пользователей
 - выдаёт готовые документы
 - управляет числом соединений
 - ограничивает пропускную способность канала
 - Кеширует готовые документы
 - Компрессирует (сжимает) выдаваемые док.
- Если документ не находится на жёстком диске, а должен быть подготовлен или извлечен из СУБД, то этим занимается CGI

CGI

- CGI = Common Gateway Interface
 - Способ взаимодействия веб-сервера с другими прикладными программами;
 - Суть консольный протокол, по которому сервер делегирует специфические запросы программам-интерпретаторам, которые на их основе подготавливают необходимые документы (или извлекают их из базы данных) и отдают их веб-серверу;
 - Для сервера считать документ по протоколу CGI так же просто, как и из файловой системы.

И последняя иллюстрация

CGI



- Вместо Python может быть любой другой серверный язык программирования: PHP, Ruby, Java...
- После User script может быть ещё один уровень – например, база данных, файловая система или запрос другой сервер

Подготовка рабочей среды

- Веб-сервер:
 - Можно проинсталлировать Apache2 себе;
 - Можно воспользоваться tallinn.mk.ee
- Текстовый редактор:
 - Поставить себе PSPad, Notepad++, EditPlus
 - Или понадеяться на редактор в компьютерном классе :)
- FTP-клиент:
 - Поставить себе FileZilla или другой клиент
 - Мучаться со встроенным в Windows cmd → ftp
 - Если используете tallinn.mk.ee + аккаунт майнора в Windows, то FTP не нужен

Fin

(анонс следующей лекции: стандарты
разметки гипертекста – SGML, (X)HTML как
основные веб-технологии)

Гипертекстовый документ

- “Форматированный” текст
- Графика
- Мультимедиа
- Ссылки на другие документы и части себя
- Интерактивные элементы
- Мета-информация

Эволюция стандартов

- 1986: SGML – Standard Generalized Markup Language
 - 1991: HTML – Hypertext Markup Language
 - 1996: XML – eXtended Markup Language
- Стандарты веба разрабатывает W3C (WWW Consortium) - группа компаний и нанятых специалистов при участии интернет-сообщества
 - w3c.org (исходные тексты)
 - w3schools.com (учебные материалы)

Пример HTML-документа

```
<HTML>
```

```
<HEAD>
```

```
<TITLE>My first HTML document</TITLE>
```

```
</HEAD>
```

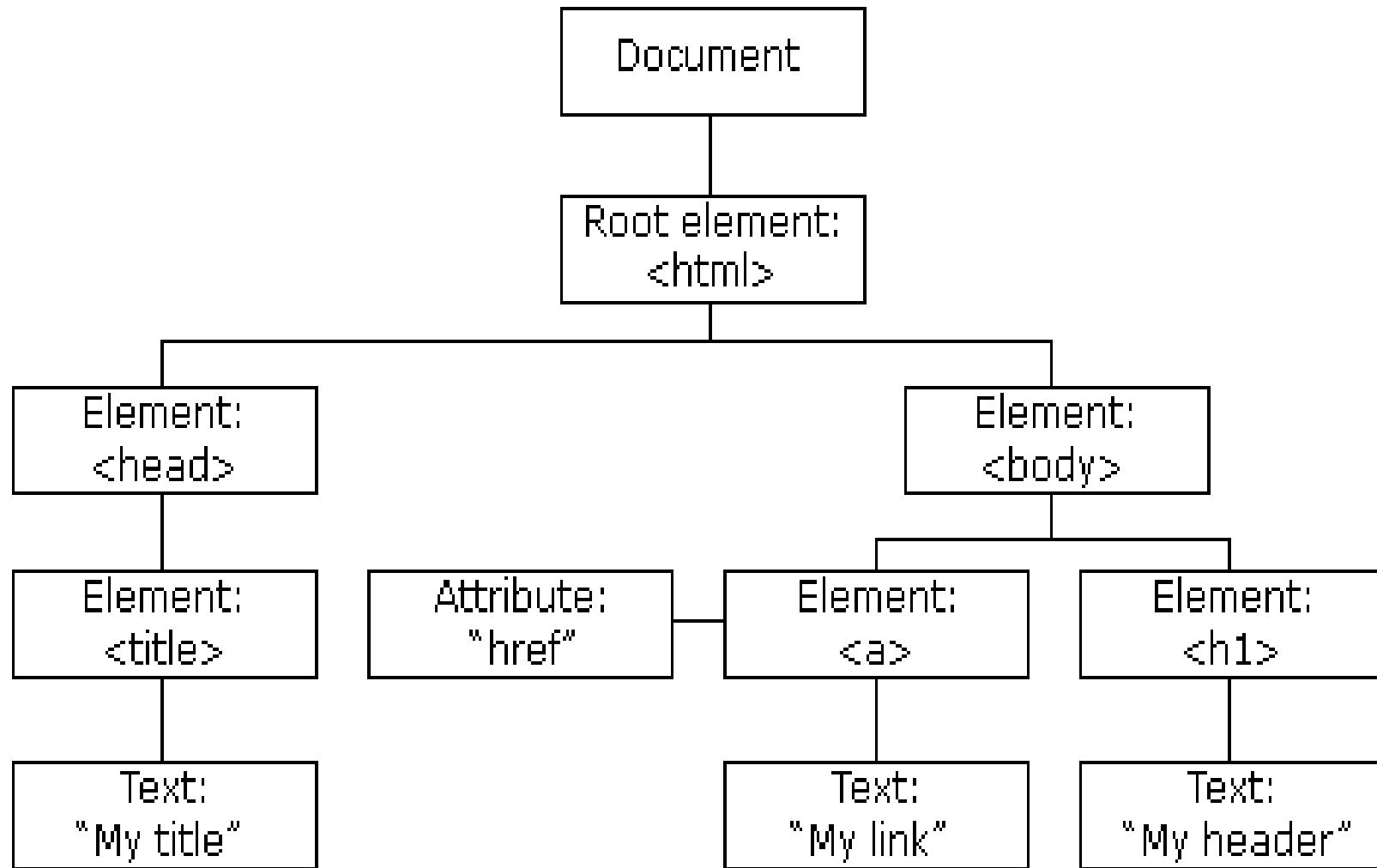
```
<BODY>
```

```
<P>Hello world!</P>
```

```
</BODY>
```

```
</HTML>
```

Структура HTML-документа



Основная задача HTML

Мы используем HTML для того, чтобы представить семантическую структуру документа!

При помощи HTML мы пытаемся объяснить браузеру (или другому обработчику) суть представленной в документе информации, а для её “раскрашивания” и добавления интерактива используем другие средства.

Знакомство с HTML

- inline и block элементы, их margin, padding, border
- Атрибуты (важные общие: id, class, lang, title, style)
 - id и class называем с буквы, а не с цифры!
- Комментарии
 - (<!-- видны всем, избегайте двойного тире -->)
- Meta-теги
 - Используем по-максимуму!
- Doctype
 - Всегда указываем правильный!
- Фреймы
 - Избегаем! В крайнем случае - <iframe>

Представление текста в HTML

- Заголовки (<h1>, <h2>)
- Контейнеры (, <pre>, <p>, <code>)
- Списки
 - Ordered vs Unordered
- Стандартное форматирование
 - vs , <i> vs , <s>, <u>
- Аббревиатуры, акронимы, цитаты
 - <q>, <cite>, <abbr>, <acronym>
- Кодировки, спецсимволы, html-entities

Ссылки в HTML

`Name`

- Абсолютные и относительные ссылки)
- Внутренние ссылки (якоря)
 - `` ``
- Active, visited и hover
- Атрибут target
 - Избегайте открывать ссылки в новом окне!
 - почему?
- Важность подчёркивания и изменения внешнего вида курсора

Изображения в HTML

```

```

- Следите, чтобы файлы были “живы”
(кстати, это относится и к ссылкам!)
- Всегда задавайте alt атрибут
- Укажите точные размеры изображения
- Не уменьшайте изображение при помощи атрибутов height и width
 - Кстати, почему?

Эти странные image maps

- Пример использования на Wikipedia

```
<img ... usemap="#map_name" />
```

```
<map name="map_name">
```

```
<area shape="rect" coords="a,b,c,d" href="link" alt="txt" />
```

```
</map>
```

- Shapes

- Rect (x1, y1, x2, y2): верхний левый, нижний правый

- Circle (x,y,radius)

- Poly (x1,y1,x2,y2,...,xn,yn): браузер замкнёт сам

- Верхний левый угол имеет координаты (0,0)

Таблицы в HTML

`<table border="x" cellspacing="y" cellpadding="z" width="%|px">`

- `<caption>`
- Блоки `<thead>`, `<tbody>`, `<tfoot>` в любом порядке
 - `<tr>` для рядов
 - `<th>` для ячеек-заголовков или `<td>` для ячеек с данными
 - Атрибуты `colspan` и `rowspan`
 - Проблема пустых ячеек

Формы

(или наконец-то что-то действительно интересное)

- Интерфейс для отправки данных на сервер
 - Хотя... не только :)
- Функционально и визуально ограничен
- Передаёт текстовые или бинарные данные
- Использует методы GET или POST
- Проектирование форм = проектирование интерфейса веб-приложения
 - Трудоёмко
 - Архиважно для веб-приложения
 - У большинства не получается :)

Элементы веб-формы

```
<form action="uri" method="post|get" enctype="">
```

```
<fieldset>
```

- <legend>

- <input type="..." name="" value="" size="" />

- Без name значение не отправится!

- <label for="id">

- <select>

- <optgroup>

- <option>

- <textarea>

Input-элементы

- Text (maxlength, readonly)
- Password (maxlength, readonly)
- Hidden
- Checkbox (checked)
- Radio (checked)
- File (accept)
- Button
- Image (alt, src)
- Submit
- Reset

Сделаем форму сами...

- Для поиска в Google
- Для отправки e-mail

CSS

Cascading Style Sheets или Каскадные страницы стилей

- Отвечает за то, как конкретный документ выглядит на конкретном устройстве.
- Введён в HTML 4
 - отделяет оформление от семантики
- Уровни (levels):
 - CSS1 (устарел)
 - CSS2 (стандарт)
 - CSS3 (недалёкое будущее, част.поддержка)
- Профили (profiles)
 - mobile, TV, print, синтезаторы речи и др.

Синтаксис CSS

- `/*` комментарии не такие, как в HTML! `*/`
- `Selector { property: value; property2: value2; }`
- Selector позволяет адресоваться к отдельным элементам (блокам) HTML-документа:
 - По html-тегу (все указанные теги)
 - `h1 { color: red; }`
 - `h1 em { color: blue; }`
 - По id (уникально к конкретному элементу)
 - `#my_id { color: blue; }`
 - По классу (все эл-ты с данным классом)
 - `.big { font-size: 150% }`
 - `p.small { font-size: 75% }`

Где разместить CSS?

- Внутри атрибута `style`, который есть почти у всех `html`-элементов (`inline`-способ)
- Внутри тега `<style>` в секции `<head>`
- Во внешнем файле (подключив в `html`-документ тегом
`<link rel="stylesheet" type="text/css" href="url" />`)

Почему надо использовать CSS во внешних файлах?

- **Удобство:** оформление отделено от разметки документа; изменяя разметку не надо переживать за целостность содержания, и наоборот.
- **Простота:** CSS настолько прост, что с его редактированием справится даже новичок
- **Производительность:** один CSS-файл можно использовать с разными документами и он будет загружаться только 1 раз (а не с каждым из документов!)

Кстати, о цветах и шрифтах

- Цвета (шрифта, фона, borders):
 - Предопределенные названия (white, red)
 - Rgb(255,255,255), Rgb(255,0,0)
 - #ffffff, #ff0000
 - #fff, f00 – можно сокращать в 2 раза
- Шрифты и отступы (margin, padding)
 - 1em = относительная величина (высота шрифта родительского эл-та, или Times New Roman 16 или другой пользовательский шрифт в настройках браузера)
 - px, pt, pc, cm, in, mm

Кстати, почему “каскадные”?

- К элементам применяется “виртуальный” стиль, который браузер соберёт, “аккумулируя” все свойства на всех уровнях: от настроек браузера до inline (последний имеет наибольший приоритет)
- Дочерний эл-т наследует свойства родителя
- Одинаковые свойства перезаписываются, новые свойства добавляются
- Если на одном уровне нужно выбрать между 2 свойствами, то будет выбрано то, что стоит ниже в файле или отмечено !important

Селекторы псевдо-классов

- Точный список зависит от уровня CSS!
- Не все браузеры поддерживают все псевдо-классы уровней 2 и 3
- `a:active`, `a:visited`, `a:hover`, `a:link`
- `:first-child`
- `:focus`
- `:lang`
- `:first-line`

Селекторы атрибутов

- **[attribute]** – установлен атрибут (значение не важно)
- **[attribute=value]** – установлен атрибут со значением, точно равным value
- **[attribute~value]** – значением атрибута является несколько слов, разделённых пробелами, и одно из них равно value
- **[attribute|=value]** значением атрибута является несколько слов, разделённых дефисами, и одно из них равно value

Элемент `<div>`

- `<div>` = division (секция, блок, раздел)
- “прозрачный” контейнер типа block
 - кстати, `` - прозрачный контейнер типа inline
- Используется для группировки нескольких (преимущественно block-типа) элементов для задания им общего стиля при помощи CSS
 - Соответственно, `` используется для если нужно задать стиль для текста или текста + нескольких inline-элементов

Свойство `display`

- Помните про `inline` и `block` элементы?
 - Это всего-навсего настройка CSS по умолчанию
 - `Display: inline;`
 - `Display: block;`
 - `Display: none;` (ой!)

Свойство position

- Position: static (по-умолчанию в общем потоке)
- Position: fixed
 - Зафиксированное положение в окне браузера (не изменяется при скролле!)
- Position: relative
 - box model остаётся в общем потоке, но её контент-часть смещается относительно своего обычного расположения
- Position: absolute
 - положение относительно первого родительского элемента с не-static позиционированием, а в его отсутствие – относительно <html> элемента. Исключается из общего потока и становится “прозрачным” для других эл-тов.

Управление position

- Для всех типов, кроме static:
 - Top
 - Bottom
 - Left
 - Right
 - z-index

Свойство float

- Сдвигать элемент в его возможное крайнее левое или правое положение
- Следующий элемент будет “обтекать” его
- Предыдущие элементы проигнорируют его
- Чтобы только сдвинуть элемент, но запретить обтекание для следующего элемента, нужно использовать `clear: both`

Box-model



CSS reset

- Полагаться на стандартные настройки браузера чревато циановым Lucydia Handwriting по магентовому фону
- Различные браузеры могут выставлять различные величины margin, padding и других свойств по-умолчанию
- Для обеспечения максимальной совместимости желательно “сбросить” все основные настройки CSS в 0 или же прописать необходимые для себя значения

Пример простейшего CSS reset

```
* {  
  vertical-align: baseline;  
  font-weight: inherit;  
  font-family: inherit;  
  font-style: inherit;  
  font-size: 100%;  
  border: 0 none;  
  outline: 0;  
  padding: 0;  
  margin: 0;  
}
```

Другой вариант CSS reset

```
html, body { padding: 0; margin: 0; }
```

```
html { font-size: 1em; }
```

```
body { font-size: 100%; }
```

```
a img, :link img, :visited img { border: 0; }
```

CSS Frameworks (каркасы)

- Предоставляют predetermined настройки и документированные договорённости для создания классической вёрстки без использования таблиц:
 - “Шапка”, гор. меню или верт. меню, 1-2-3 колонки, “подвал”
- Обеспечивают хорошую кросс-браузерную совместимость
- Максимально соответствуют спецификациям W3C

Популярные CSS каркасы

- Blueprint CSS framework
- YUI grids
- YAML
- Elastic
- 960 Grid System
- Baseline CSS framework

Валидаторы HTML и CSS

- Quality Assurance Tools на сайте W3C
 - <http://www.w3.org/QA/Tools/>
- Валидный HTML-документ может не обеспечивать кросс-браузерной совместимости (пример: <http://acid3.acidtests.org/>)
- Максимально валидный HTML-документ в первую очередь говорит о компетенции разработчика, упрощает переносимость кода, его развитие и передачу в пользование другим разработчикам

Веб-программирование

В программе

- Javascript
 - (12 часов, 2 дня x 6 часов)
- jQuery
 - (6 часов, 1 день)
- XML, JSON и AJAX
 - (6 часов, 1 день)
- PHP
 - (24 часа, 4 дня x 6 часов)
- Домашние работы (3-4) и экзамен (программирование на PHP+Javascript)

JavaScript

- Скриптовый – выполняется в веб-браузере на стороне клиента
- Интерпретируемый
- Бесплатный – лицензия не нужна
- Относительно безопасный – работает в “песочнице” браузера, изолирован от операционной системы и файловой системы компьютера
- Кросс-платформенный – хорошо поддержан всеми браузерами, код в целом совместим

Эволюция Javascript

- Создан в 1995-1997 году организацией ECMA, другой название: ECMAScript
- Родственники: ActionScript (Adobe Flash) и Jscript (MS .NET & Windows Script Engine)
- Кроме браузеров используется в Adobe Acrobat, OpenOffice и других приложениях
- Современные браузеры поддерживают версию Javascript 1.5 (Jscript 6 в IE8), хотя существует версия 1.8; активно развивается.

Javascript != Java

- Не путайте Java и Javascript (это первый признак дремучего ламера :)
- Java – ЯП общего назначения (программа исполняется из прекомпилированного байткода, выступает в одной “весовой категории” с C++)
- Оба языка имеют похожий синтаксис (наследован от C) и схожий набор “стандартных библиотек” (набор встроенных функций) – знание одного ЯП упрощает изучение другого, но не более того.

Что может Javascript?

- Определить пользовательский браузер и его настройки (размер экрана и т.п.)
- Динамически изменять HTML-страницу (добавлять, удалять, редактировать, прятать и показывать элементы)
- Обработать события, поступающие от пользователя (реагировать на клик, перемещение мыши, нажатия клавиш и т.п.)
- Проверять формы перед отправкой
- Выполнять действия по таймеру
- Устанавливать и читать cookies

Для чего нам Javascript?

- Снять часть нагрузки с веб-сервера
 - Не заставлять сервер обрабатывать неправильно или не полностью заполненные формы
 - Клиентские компьютеры достаточно мощные для того, чтобы самостоятельно обрабатывать значительные объёмы данных (сортировка, фильтрование и т.п.)
- Обойти проблемы с совместимостью HTML и CSS между браузерами
- Создавать более богатые, функциональные и удобные пользовательские интерфейсы

Ограничения на использование Javascript

- Шифрование, проверка паролей и прав (всё, что связано с безопасностью - нельзя)
- Если ваша целевая аудитория – старые браузеры и медленные компьютеры или браузеры без поддержки JS (выключен у параноиков; не поддерживается некоторыми встраиваемыми или текстовыми браузерами)
- Если вам нужно создавать анимацию (Flash)

Техническая характеристика

- Расширение файлов: .js
- Императивный (структурный) ЯП
- Объектно-ориентированный (почти :)
- Динамическая типизация переменных
- Поддерживает регулярные выражения
- Регистро-зависимый
- Разделитель выражений (;) может быть опущен, если его отсутствие не вводит интерпретатор в заблуждение

Инструментарий разработчика

- Ваш любимый текстовый редактор (с подсветкой синтаксиса)
- Add-on Firebug для Firefox
- Для Chrome - встроенные в Developer Tools
- Для IE и других браузеров... лучше использовать Firefox или Chrome :)

Hello, world!

```
<script>
```

```
    document.write('Hello, world!');
```

```
    alert('Javascript executed!');
```

```
</script>
```

Облагородим код

```
<script type="text/javascript">  
  <!--  
  /* код выполнится только в браузерах,  
  поддерживающих Javascript */  
  
  document.write('Hello, world!');  
  alert('Javascript executed!');  
  
  //-->  
</script>
```

Pop-up boxes

Для отладки программ удобно использовать:

- `Alert('text');`
- `Confirm('text');` //true или false
- `Prompt('text','default')`

Где разместить Javascript?

- Внутри некоторых атрибутов:
`Click me`
- Внутри тега `<script>` в секции `<body>`
 - Исполняются при загрузке страницы
- Внутри тега `<script>` в секции `<head>`
 - Выполняются по вызову, должны быть оформлены как функции
- Во внешнем файле, подключив в html-документ тегом
`<script type="text/javascript" src="script.js"></script>`

Переменные 1/2

- Начинаются с буквы или подчёркивания
- Регистрозависимы: `x` и `X` – разные!
- Не имеют заранее определенного типа и хранят то, что в них запишут
- Могут менять тип по ходу выполнения (`1` как число и `"1"` как строка)
- Иницируются присвоением значения в любом месте скрипта
- Можно инициировать явно при помощи `var` (получаются т.н. “глобальные переменные”)
- Реинициализация не стирает значение!

Переменные 2/2

- Массивы

```
var friends = new Array();
```

```
friends[0] = "Вася"; // нумерация с 0
```

```
friends[1] = "Петя";
```

```
friends[2] = "Маша";
```

- Бинарные: true и false

- Строковые: тип кавычек не имеет значения, внутренние кавычки того же типа экранируются слешем, спецсимвол \n

- Неопределенные: NaN и undefined

Функции 1/2

- Объединяют блок кода, решающий некую задачу
- Служат для повторного использования кода
- Предотвращают автовыполнение кода
- Позволяют конструировать “чёрные ящики” (передаём несколько параметров, получаем результат)
- Лучше помещать в `<head>` или внешний файл (внутри `<body>` нужно определять функции прежде, чем их использовать)

Функции 2/2

```
function product(a,b) {  
    return a*b;  
}
```

```
a = 2;
```

```
b = 3;
```

```
x = product(a,b);
```

```
alert(x);
```

Операторы

- Арифметические: + - * / %
 - Унарные: - + (изм. знача; изм. Типа)
 - Инкрементальные: ++ и – (перед и после)
- Сравнение: < > <= >= == === != !==
 - == игнорирует тип переменных
 - === учитывает тип переменных
- Логические: ! && ||
- Тернарный: (test) ? (true) : (false)

Управляющие конструкции и циклы

- If...else
- Switch
- For и For...in
- Do...while и While
- Break и Continue

Событийная модель (events)

- Почти каждый элемент в DOM (и все HTML-элементы на странице) могут реагировать на происходящие с ними события, например:
 - OnLoad
 - OnFocus / OnBlur
 - OnSubmit
 - OnMouseOver / onMouseOut
 - OnKeyUp / onKeyDown
 - OnMouseUp / onMouseDown

Обработка ошибок выполнения

```
Try {
```

выполняем код, который может привести к ошибке выполнения – например всё, что связано с пользовательским вводом или кросс-браузерной совместимостью

```
    throw “имя_ошибки”;
```

```
} catch (e) {
```

```
    if(e == “имя_ошибки”) { обрабатываем }
```

```
}
```

Практическая часть

- Управление фокусом
- Валидация формы
- Анимация подменой картинок
- Управление видимостью объектов
- Определение устаревших браузеров
- Раскрашивание рядов таблицы “зеброй”
- Крестики-нолики
- Сортировка таблицы

Контроль над эл-тами DOM 1/2

- `document.getElementById("some_id")`
 - Возвращает искомый Объект DOM
- `dom_object.getElementsByTagName("tag")`
 - Возвращает набор (*collection*) элементов
 - Collection это Array, применительно к объекту
 - Мы можем перебирать эл-ты набора `for`

Контроль над эл-тами DOM 2/2

- `dom_node.className`
 - считать/установить атрибут `class`
- `dom_node.style.styleName`
 - Только для свойств CSS, заданных `inline`
- `dom_node.innerHTML`
 - считать/установить содержимое объекта DOM в виде HTML (т.е. в виде текста, а не объектов)

Основные объекты стандартной библиотеки

- window (он же self)
 - Корневой объект объектной модели JS
 - alert() по сути метод window.alert()
 - В большинстве случаев опускаем (не пишем)
 - history.
 - location.
 - Math.
 - navigator.

Объект history

- `history.back()`
- `history.forward()`

Объект location

- Location.
 - hash (от # и дальше)
 - hostname
 - href (весь URL как он есть)
 - pathname
 - search (от ? И дальше)
 - reload([bool forceFullReload])
 - replace(string location)

Объект Math

- Math.
 - [константы] – E, LN2, LN10, PI, SQRT2
 - [тригонометрия] – sin(), cos(), tan()
 - max(n1, n2, n3, ..., nN), и такой же min()
 - round(), floor(), ceil() - до целого
 - pow(n, k) – n в степени k
 - random() - на интервале 0..1

Объект navigator

- navigator.
 - appName
 - appVersion
 - language (Moz) или userLanguage (IE)
 - platform
 - appCodeName
 - userAgent
- Можем полагаться только на честность пользователя, т.к. все эти атрибуты ненадёжны и могут содержать что угодно

Полезные ссылки

- [Javascript tutorial @ HowToCreate.co.uk](http://HowToCreate.co.uk)
- [Javascript tutorial @ W3Schools.com](http://W3Schools.com)
- Примеры манипуляции элементами DOM

jQuery

- Каркас (framework) для более
 - Быстрого
 - Кроссбраузерного
 - Наглядного

использования Javascript

- <http://jquery.com/>
- 1 файл (сейчас – jquery-1.4.2-min.js)
- И репозиторий плагинов на все случаи жизни: <http://plugins.jquery.com/>

Подключаем JQuery

```
<script type="text/javascript" src="jquery-1.4.2.min.js"></script>
```

И добавим плагин Colorize

<http://plugins.jquery.com/project/Colorize>

```
<script type="text/javascript" src="jquery.colorize-2.0.0.js" ></script>
```

Пример использования jQuery

- Из консоли:

```
$("#factsheet").colorize( );
```

где #factsheet – ID таблицы

- Из кода страницы:

```
$(document).ready(function() {
```

```
  $('#aactsheet').colorize();
```

```
});
```

Что есть что в jQuery

- `$()` = сокращение для объекта jQuery
- Параметры ф-ции `$()` – селекторы DOM, определяющие объекты для произведения дальнейших действий. Как правило, возвращают массив элементов или 1 эл-т

`$('a')` возвращает коллекцию эл-тов `<a>`

- Методы `$()` – действия, которые будут выполнены надо всеми (или реже – только над первым) элементами, указанными селектором.

`$('a').size()` возвращает число эл-тов `<a>`

Селекторы jQuery

- Похожи на те, что мы использовали в CSS:

`$('element')`

`$('.class')`

`$('#id')`

- Можно перечислить несколько селекторов:

`$('a, p, h1')`

- Выбрать все элементы: `$('*')`

- Использовать фильтры:

`$(":button"), $(':checked'), $(':disabled')`

Более сложные селекторы

- `$('.ul.class > li')` все `` внутри `<ul class="class">`
- `$("td:empty")` все “пустые” `<td></td>`
- `$("a:eq(2)")` третий (считаем с 0!) по счёту `<a>`
- `$("p:contains('Hello'))` ищет `<p>Hello, world!</p>`
- `$("label + input")` `<input>`, перед которым `<label>`
- `$("#group ~ p")` все `<p>` после `<... id="group">`, но не вложенные в него!

Чем можно управлять?

- `$('#selector').addClass('classname')`
- `$('#selector').attr('src', 'image.jpg')`
- `$('#selector').html('Hello, world!')`
- `$('#selector').text('Hello, world!')`
- `$('#selector').css('background-color', '#666')`
- `$(this).hide("slow");`

Получение значений

- Зачастую одни и те же методы являются и setter-ами (установщиками новых значений), и getter-ами (возвращают текущее значение)
 - `var src = $('selector').attr('src')`
 - `var content = $('selector').html()`
 - `var txt = $('selector').text()`
 - `var h = $('selector').height()`

Обработчики событий

```
$(document).ready( function() {  
    $("a").click( function(event) {  
        alert("Links do not work anymore!");  
        event.preventDefault();  
    });  
});
```

Другие особенности

- Последовательное выполнение методов (*chaining*):

```
$('#selector').hide().addClass('hidden')
```

- Определение браузера:

```
if ($.browser.msie) {  
    // code for Internet Explorer  
}
```

PHP

- PHP: Hypertext Preprocessor
- Интерпретируемый скриптовый ЯП
- Выполняется на стороне сервера
- Универсальный ЯП для веб-разработки, но может быть использован и в других целях
- Простой, быстрый, распространённый, дешёвый в обслуживании = популярный
- История с 1995 года; свежая версия – 5.3
- Офдок: <http://php.net/>

Техническая характеристика

- Мультиплатформенный (Windows, Linux, etc)
- Написан на C – высокопроизводительный
- Объектно-ориентированный (с v5)
- Динамические, регистрозависимые переменные
- Нет полной поддержки Unicode! (обещают в v6)
- Использование разделителя (;) - обязательно в конце каждого предложения!
- /* комментарий многострочный */ //или так
- Отличительная черта – \$ перед именем переменной и размещение кода между тегами `<?php ?>`

Принцип работы

Web-browser	
Web-server	DBMS
PHP	
Op.system	

- PHP в вебе является конструктором веб-страниц
- Цель: обработать полученные параметры GET/POST запроса и “собрать” HTML-документ, который требует клиент
- Позволяет автоматизировать создание HTML-документа, включить в него данные из СУБД, файловой системы и т.п.

Что можно сделать на PHP?

- CMS (Content Management Systems) – Joomla, Drupal, Wordpress, Wiki
- Инфосистемы практически любого уровня сложности (Moodle)
- Социальные сети (Livejournal, Facebook)
- Решить практически все задачи, стоящие перед веб-разработчиком:
авторизация/аутентификация
пользователей, работа с сессиями и cookies,
раюота с различными типами файлов,
СУБД, XML, почтовыми службами и т.п.

Базовый принцип

- Было:

```
<p>Hello, world!</p>
```

- Стало:

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

- То же самое, но сокращённо:

```
<p><?='Hello, world!'?></p>
```

Создание динамического содержимого

```
<?php
```

```
    $greeting = "Hello, world!";  
    echo $greeting;
```

```
?>
```

дальше в любом месте кода:

```
<?=$greeting?>
```

- NB! Судьба “short_open_tag” в PHP v6 неизвестна, поэтому лучше использовать `<?php ... ?>`

Базовый синтаксис

- + - * /
- == != > < >= <=
- && ||
- Конкатенация (склеивание) строк - . (точка)

```
<? $object = 'world';  
    echo "Hello, " . $object . "!"; ?>
```
- For, if..else, switch, do-while как в JS, C#, C
- ```
<? function greet($obj) { echo "Hello, ".$obj."!"; }
 greet("world"); greet("Zahhar"); ?>
```

# Массивы

```
<?php $likes = array('Juice', 'Beer', 'Meat'); ?>
```

```

```

```
<?php foreach($likes as $item) { ?>
```

```
<?php echo $item; ?>
```

```
<? } /* foreach ends here! */ ?>
```

```

```

# Ассоциативные массивы

```
<?php
 $links = array(
 array(
 'url' => 'http://www.mk.ee',
 'text' => 'Mainori kõrgkool'
),
 array(
 'url' => 'http://www.livejournal.com',
 'text' => 'My blog'
)
); ?>
```

# Вывод ассоциативного массива

```

```

```
<?php foreach($links as $link) { ?>
```

```

```

```
<a href="<?php echo $link['url']; ?>">
```

```
<?php echo $link['text']; ?>
```

```

```

```

```

```
<? } /* foreach ends here! */ ?>
```

```

```

# Параметры

/index.php?name=Zahhar

```
<p><?php echo $_GET['name']; ?></p>
```

**NB! Никогда так не делайте!** Security issue: пользовательский ввод необходимо строго проверять перед внедрением в страницу, иначе – XSS, SQL Injection и прочие хаки!

- Используйте `strip_tags()` и `htmlspecialchars()`
- Для чисел проверяйте диапазон значений, для строк – `strlen()`, `substr()`, `trim()`

# Подключение внешних файлов

```
<?php include 'header.html'; ?>
```

```
<?php include 'navigation.php'; ?>
```

Content

```
<?php include 'footer.html'; ?>
```

```
<?php include 'sidebar.html'; ?>
```

# XML

- XML (Extensible Markup Language) – подмножество языка SGML (как и HTML), предназначенный для структурированной разметки любых документов
- Содержит только данные и инструкции, описывающие их взаимосвязь и структуру
- Не содержат информации о семантике (значении) или способе отображения (визуализации) данных

# XML

- XML – текстовый документ, синтаксис которого схож с синтаксисом HTML
- Обязательный заголовок:  

```
<?xml version='1.0' encoding='UTF-16' standalone='yes' ?>
```
- Список тегов (и атрибутов) не стандартизирован:  

```
<myNewTag myProperty="whatever">
 some data </myNewTag>
```
- Закрывающие теги обязательны!

# XML

- XML-документы могут содержать ссылки на другие документы
- При помощи XML можно создать свой язык разметки (набор разрешенных тегов и атрибутов, правила по их использованию)
- Проверка синтаксиса производится по шаблонам DTD или XMLSchema, на которые должна быть ссылка в начале XML-документа
- Сам шаблон должен быть доступен парсеру, проверяющему валидность документа

# Регулярные выражения

- Регулярные выражения (regular expressions) – это технология, позволяющая искать данные в текстовых строках на основе заданного шаблона соответствия
- Например:
  - Найти на сайте все адреса эл.почты  
`/^\w+([\.\w]+)*\w@\w((\.\w)*\w+)*\.\w{2,3}$/`
  - Найти все “личные коды” (isikukood)
  - Найти всё содержимое тегов `<b>...</b>`
  - Найти все слова, с повторяющимися согласными (троллейбус, коллизия)

# Когда использовать `gedexr`?

- Если у вас нет точного представления о природе данных
- Если искомые вами данные лучше описываются шаблоном, чем перечислением конкретных значений
- Если вы желаете сократить число строк кода
- Если вам нужно обработать большое кол-во слабоструктурированных данных

# В чём минусы regex?

- Работают медленнее, чем операции со строками
- Делаю код менее читабельным и нуждающимся в комментариях
- Синтаксис регулярных выражений может несколько меняться в различных диалектах и языках программирования

# Ajax

- AJAX = **A**sync. **J**avascript **a**nd **X**ML (2005)
- Сейчас модно писать Ajax :)
- Это не отдельная технология, а набор технологий (DOM, CSS, HTTP Request)
- Суть: избавить пользователя от постоянного обновления (refresh) экрана браузера при работе с веб-приложениями
- Эффект: субъективное ощущение, что приложение работает быстрее и “плавнее”, подобно десктопному приложению
- Выгода: позволяет ровнее распределить нагрузку на веб-сервер

# Состав Ajax

- CSS & XHTML для представления данных
- XMLHttpRequest object (компонента браузера) для выполнения запроса
- XML как стандарт передачи и манипуляции данными
- DOM для управления элементами страницы
- Javascript чтобы заставить всё работать вместе

# Поддержка Ajax

- Все современные браузеры включают в себя компонент XMLHttpRequest object
  - Firefox 1.0 and up
  - Opera 7.6 and up
  - Safari 1.2 and up
  - Chrome
  - MSIE 7 and up
- MSIE 5.5 и 6.0 вместо него включают ActiveX компоненты Microsoft.XMLHTTP и Msxml2.XMLHTTP соответственно

# XMLHttpRequest

- readyState – Number (0,1,2,3,4)
- .responseText – String
- .responseXML – Document
- Status – Number
- StatusText – String
- Onreadystatechange – Event Listener (function)

# XMLHttpRequest

- Void open (method, uri, async)
- Void send (data)
- Void abort ()
- Void setRequestHeader(name, value)
- String getResponseHeader(name)
- String getAllResponseHeaders()

# Ограничение Ajax

В целях безопасности  
Ajax ограничен  
только тем доменом,  
в котором он был  
инициализирован

Мы не можем сделать  
Ajax-запрос из `mk.ee` на `google.com`

# Ајах и jQuery

- JS фреймворки позволяют не беспокоиться о поддержке браузером Ајах, если включена поддержка JS
- Плагины для JS фреймворков позволяют выполнять cross-domain Ајах-запросы
- Для обмена данными может быть использован не XML, а JSON или ряд других стандартов