Web application frameworks

Ralf Lämmel
Software Languages Team
University of Koblenz-Landau

Acknowledgement: This slide deck also uses contributions by Andrei Varanovich.
Introduction
Web application

A web application or web app is any program that runs in a web browser. It is created in a browser-supported programming language (such as the combination of JavaScript, HTML and CSS) and relies on a web browser to render the application.[1][2][3]

Web applications are popular due to the ubiquity of web browsers, and the convenience of using a web browser as a client to update and maintain web applications without distributing and installing software on potentially thousands of client computers is a key reason for their popularity, as is the inherent support for cross-platform compatibility. Common web applications include webmail, online retail sales, online auctions, wikis and many other functions.
A Web application (WA) is a software system based on technologies and standards of the WWW that provides Web-specific resources such as content and services and is accessed via a user interface -- the Web browser.

Principle architecture of WA

- It is essentially a Client-server architecture.
- End users access a *client* (in a Web browser).
- Services are implemented on a (Web) *server*.
- Services are materialized as *documents*.
- Client & server cooperate via *protocols* over the internet.

supports rendering and interaction with HTML pages

delivers Web pages over HTTP

describe message formats and rules for exchanging messages
WA architecture

What if the client does not need a server all the time?

What if the server is often unavailable?

What’s the request?

What if this is a native app?

Is that a complete view, a delta, or what?

What if this is a Web service?
Demo

A Web application for a simple human-resources management system.

http://101companies.org/wiki/Contribution:html5local

No internals are discussed at this point. Appearance and architecture options are discussed.
A web application framework (WAF) is a software framework that is designed to support the development of dynamic websites, web applications, web services and web resources. The framework aims to alleviate the overhead associated with common activities performed in web development. For example, many frameworks provide libraries for database access, templating frameworks and session management, and they often promote code reuse.[1] For a comparison of concrete web application frameworks, see Comparison of web application frameworks.

4 Features

4.1 Web template system
4.2 Caching
4.3 Security
4.4 Database access, mapping and configuration
4.5 Scaffolding
4.6 URL mapping
4.7 Ajax
4.8 Web services
4.9 Web resources
Pages in category "Web application frameworks"

The following 172 pages are in this category, out of 172 total. This list may not reflect recent changes (learn more).

- List of Ajax frameworks
- Microframework
- Web application framework
- Comparison of web application frameworks

L
- Laravel
- Lift (web framework)
- Lithium (PHP framework)
- Lively Kernel
- LYME (software bundle)

A
- ActiveVFP
- Agavi
- AIDA/Web

M
- Mach-II
- Makumba (framework)
Theme: Web programming

Headline

Demonstrations of web programming

Metadata

- Contribution: hoppstack  memberOf  this
- Contribution: html5ajax  memberOf  this
- Contribution: html5local  memberOf  this
- Contribution: jsf  memberOf  this
- Contribution: pyjamas  memberOf  this
- Contribution: rubyonrails  memberOf  this
- Contribution: seaside  memberOf  this
- Contribution: silverlight  memberOf  this
- Contribution: strutsAnnotation  memberOf  this
- Contribution: zend  memberOf  this
What are the characteristics of Web applications and their development?
Rich functionality
Heterogenous resources
Secure operation
Responsive user interface
Slick appearance
Support for different browsers

http://www.unixmen.com/browser-war-heats-up-again-as-chrome-unseats-ie-for-may-2012/
Support for different versions

http://arstechnica.com/business/2012/02/2001-all-over-again-internet-explorer-6-share-grows-and-chrome-falls/
Support for different platforms
Description

Instagram – A beautiful way to share your world. It's fast, free and fun!

Pick from one of several gorgeous filtered effects to breathe a new life into your mobile photos. Transform everyday moments into works of art you'll want to share with friends and family.

Share your photos in a simple photo stream with friends to see - and follow your friends' photos with the click of a single button. Every day you open up Instagram, you'll see new photos from your closest friends, and creative people from around the world.

Visit Developer's Website ›  Email Developer ›  Privacy Policy ›

App Screenshots
Instagram
By Burbn, Inc.

Open iTunes to buy and download apps.

Description
★★★★★ Instagram

80 million users love Instagram! It's a free, fun, and simple way to make and share gorgeous photos on your

Burbn, Inc. Web Site › Instagram Support ›

What's New in Version 3.0.0
- View your photos on a map (visit your profile and tap Photo Map to select which of your photos will be viewable on the map)
- Redesigned profile screen

iPhone Screenshots

© 2011 Burbn, Inc.
Rated 12+ for the following:
Infrequent/Mild Sexual Content or Nudity
Infrequent/Mild Profanity or Crude Humor
Infrequent/Mild
Mature/Suggestive Themes
Infrequent/Mild Alcohol, Tobacco, or Drug Use or References
Reliance on browser plug-ins
Get Silverlight 5

Silverlight is a powerful development tool for creating engaging, interactive user experiences for Web and mobile applications. Silverlight is a free plug-in, powered by the .NET framework and compatible with multiple browsers, devices and operating systems, bringing a new level of interactivity wherever the Web works.

DOWNLOAD NOW
Adobe AIR 3.4 (23.4 MB)

Your system: Macintosh, English
Different operating system?

Learn more | System requirements | Distribute Adobe AIR

Download now

You may have to temporarily disable your antivirus software.
By clicking the Download now button, you acknowledge that you have read and agree to the Adobe Software Licensing Agreement.
Reliance on cross-platform tools

```html
<html>
<head>
<title>HelloWorld Application</title>
</head>
<body>
<p>Hello, World!</p>
</body>
</html>

public class HelloWorldApplication {

    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Recommended resources: ‘HTML5 for the Java developer’
http://www.youtube.com/watch?v=l-GD4ro9fh4
http://www.youtube.com/watch?v=2msljsQcls
Anything else you want to emphasize as a characteristic?
Basics of web-application development
Good old times

$ telnet www.microsoft.com 80
Trying 65.55.57.27...
Connected to lb1.www.ms.akadns.net.
Escape character is '\^]'.
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Cache-Control: no-cache
Content-Length: 1020
Content-Type: text/html
Last-Modified: Mon, 16 Mar 2009 20:35:26 GMT
Accept-Ranges: bytes
ETag: "67991fbd76a6c91:0"
Server: Microsoft-IIS/7.5
X-Powered-By: ASP.NET
Date: Fri, 07 Sep 2012 04:45:28 GMT
Connection: close

Connection closed by foreign host.
Telnet

Telnet is a network protocol used on the Internet or local area networks to provide a bidirectional interactive text-oriented communications facility using a virtual terminal connection. User data is interspersed in-band with Telnet control information in an 8-bit byte oriented data connection over the Transmission Control Protocol (TCP).

http://en.wikipedia.org/wiki/Telnet
29 June 2012
Hypertext Transfer Protocol


- GET: Request representation for resource
- HEAD: Like GET but without response body
- PUT: Upload representation for resource
- POST: Submit data for resource
- OPTIONS: Query for available methods
- CONNECT: Facilitate SSL-encrypted communication
- DELETE: Delete specified resource
- TRACE: Return request as it arrived at server
- PATCH: Partial modification of resource

© 2014-2015, Ralf Lämmel, rlaemmel@acm.org
Hypertext Transfer Protocol

- Safe: HEAD, GET, OPTIONS, TRACE
- Unsafe: POST, PUT, DELETE
- Idempotent: PUT, DELETE + safe methods
- HTTP is a stateless protocol.

We skip the underlying TCP/IP.

More on HTTP, when we talk about ‘REST’.
Static web pages


localhost/helloWorld.html

Hello World!

<html>
<head>
<title>Hello World WebApp</title>
</head>
<body>
<p>Hello World!</p>
</body>
</html>
Dynamic web pages


```perl
#!/usr/bin/perl
print "Content-type: text/html\n\n";
print "\n<html>
<head>
<title>Hello World WebApp</title>
</head>
<body>
<p>Hello World!</p>
</body>
</html>
exit;
```

localhost/helloWorld.cgi

Hello World!

HTML markup
Perl

Influenced by
- AWK, Smalltalk 80, Lisp, C,
- C++, sed, Unix shell,
- Pascal

Influenced
- Python, PHP, Ruby,
- ECMAScript, LPC, Windows
- PowerShell, JavaScript,
- Falcon, Perl 6, Qore

Implementation language
- C

OS
- Cross-platform

License
- GNU General Public
- License or Artistic
- License[3]

Usual filename extensions
- .pl .pm .t

Typing discipline
- Dynamic

Appeared in
- 1987

Designed by
- Larry Wall

Developer
- Larry Wall

Stable release
- 5.16.1[1] (August 8, 2012; 29 days ago)

Preview release
- 5.17.2[2] (July 20, 2012; 48 days ago)
CGI - Common Gateway Interface

http://www.citycat.ru/doc/CGI/overview/overview.html
The Common Gateway Interface (CGI) is a standard (see RFC 3875: CGI Version 1.1) method for web server software to delegate the generation of web content to executable files. Such files are known as CGI scripts; they are programs, often stand-alone applications, usually written in a scripting language.


We do not go into detail here. There is, for example, also the advancement **FastCGI**.
Access to server-side resources

localhost/showFile.cgi

Perl code

#!/usr/bin/perl
open FILE, "demo.txt";
my $lines = <FILE>;
print "Content-type: text/html\n\n";
print <<HTML;
<html>
<head>
<title>Spying WebApp</title>
</head>
<body>
<p>
HTML
print $lines;
print <<HTML;
</p>
</body>
</html>
HTML
exit;
HTTP request parameters

```
#!/usr/bin/perl

local ($buffer, @pairs, $pair, $name, $value, %FORM);

# Read in text
$ENV{REQUEST_METHOD} =~ tr/a-z/A-Z/;
if ($ENV{REQUEST_METHOD} eq "GET")
{
    $buffer = $ENV{'QUERY_STRING'};
}

# Split information into name/value pairs
@pairs = split(/&/, $buffer);
foreach $pair (@pairs)
{
    ($key, $val) = split(=//, $pair);
    $val =~ tr/+/%/;
    $val =~ s/%(.)/pack("C", hex($1))/eg;
    $FORM{$key} = $val;
}
$name = $FORM{name};

print "Content-type:text/html\r\n\r\n";
print "<html>";
print "<head>";
print "<title>HelloName WebApp</title>";
print "</head>";
print "<body>";
print "<h2>Hello $name!</h2>";
print "</body>";
print "</html>";
```

© 2014-2015, Ralf Lämmel, rlaemmel@acm.org
HTML forms

http://en.wikipedia.org/wiki/Form_(web)

```
<FORM action="helloName.cgi" method="GET">
Name: <input type="text" name="name">
<br>
<input type="submit" value="Submit">
</FORM>
```
Code embedded into HTML

```html
<html>
<head>
    <title>HelloWorld WebApp</title>
</head>
<body>
<?php echo '<p>Hello World!</p>'; ?>
</body>
</html>
```
**PHP - The PHP Hypertext Processor**

http://en.wikipedia.org/wiki/PHP

---

**PHP**

![PHP Logo](image)

**PHP: Hypertext Preprocessor**

<table>
<thead>
<tr>
<th>Paradigm(s)</th>
<th>imperative, object-oriented, procedural, reflective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appeared in</td>
<td>1995; 17 years ago[1]</td>
</tr>
<tr>
<td>Designed by</td>
<td>Rasmus Lerdorf</td>
</tr>
<tr>
<td>Developer</td>
<td>The PHP Group</td>
</tr>
<tr>
<td>Stable release</td>
<td>5.4.6 (August 15, 2012; 23 days ago)</td>
</tr>
<tr>
<td>Typing discipline</td>
<td>Dynamic, weak</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Major implementations</th>
<th>Zend Engine, Phalanger, Quercus, Project Zero, Hiphop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Influenced by</td>
<td>C, C++, Java, Perl, Tci[1]</td>
</tr>
<tr>
<td>Implementation language</td>
<td>C</td>
</tr>
<tr>
<td>OS</td>
<td>Cross-platform</td>
</tr>
<tr>
<td>License</td>
<td>PHP License</td>
</tr>
<tr>
<td>Usual filename extensions</td>
<td>Common extensions</td>
</tr>
<tr>
<td></td>
<td>.php</td>
</tr>
<tr>
<td></td>
<td>Older, now uncommon extensions</td>
</tr>
<tr>
<td></td>
<td>.phtml, .php4 .php3, .php5, .phps</td>
</tr>
<tr>
<td>Website</td>
<td><a href="http://www.php.net">www.php.net</a></td>
</tr>
</tbody>
</table>

[PHP Programming at Wikibooks](http://www.php.net)

---

© 2014-2015, Ralf Lämmel, rlaemmel@acm.org
Summary of basics

You learned about ...

• the difference between static and dynamic pages,
• the use of HTML forms,
• the architecture of CGI,
• basic techniques of writing CGI scripts,
• environment variables in CGI programs,
• and some bits of Perl and PHP.
Server-side Scripting
Server-side scripting is a technique used in website design which involves embedding scripts in an HTML source code which results in a user's (client's) request to the server website being handled by a script running server-side before the server responds to the client's request.

What is scripting?

• Embed simple code in HTML pages.
• The HTML pages use the code to decide what elements and data to display.
• Existing APIs can be invoked to compute information for inclusion in the web page.
Key components of Server-side scripting

• **Web Server** -- makes available documents and handles requests.

• **Server-side processor** -- interprets scripts and generates HTML
Web server can refer to either the hardware (the computer) or the software (the computer application) that helps to deliver Web content that can be accessed through the Internet.

Apache web server

- Apache http server project
- http://httpd.apache.org
- Apache foundation started to support the web server project, but eventually extended to a multitude of other projects.
Configuration of Apache to work with PHP:

Configure directory DocumentRoot

This is the directory from which the Web server will serve the documents (html, php, etc). That is, by default, requests are resolved to this directory.

Configuration entry:

DocumentRoot "/Library/WebServer/Documents"
What happens when we open a PHP file in the browser?

Non-processed HTML is shown unless PHP is enabled. `php5_module` should be enabled to handle *.php

LoadModule php5_module libexec/apache2/libphp5.so
Remember --
Code embedded into HTML

localhost/helloWorld.php

Hello World!

HTML with PHP:

```html
<html>
<head>
  <title>HelloWorld WebApp</title>
</head>
<body>
  <?php echo '<p>Hello World!</p>'; ?>
</body>
</html>
```
PHP is a general-purpose server-side scripting language originally designed for Web development to produce dynamic Web pages. It is one of the first developed server-side scripting languages to be embedded into an HTML source document rather than calling an external file to process data. The code is interpreted by a Web server with a PHP processor module which generates the resulting Web page.

PHP Flow

HTML is substituted for PHP code

Raw .php source

Request for .php file

Browser

Internet

Web Server

PHP Pre-processor

Passed source file
Adding interaction

Welcome Andrei!
You are 27 years old.

<html>
<body>
Welcome Andrei!<br/>
You are 27 years old.
</body>
</html>
intro.html

```html
<html>
<body>

<form action="welcome.php" method="post">
Name: <input type="text" name="fname" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>

</body>
</html>
```

welcome.php

```html
<html>
<body>

Welcome <?php echo $_POST["fname"];?>!<br />
You are <?php echo $_POST["age"];?> years old.

</body>
</html>
```
<?php

$lines = preg_split('/\s*\[\r\n]+\s*/', file_get_contents('prod.txt'));

foreach($lines as $i => $line) {
    $pairs = explode(';', $line);
    foreach($pairs as $pair) {
        list($column, $value) = explode('=', $pair, 2);
        $columns[$column] = true;
        $rows[$i][$column] = $value;
    }
}
$columns = array_keys($columns);
echo '<table><thead><tr>';
foreach($columns as $column) {
    echo '<th>'.$column.'</th>';
}
echo '</tr></thead><tbody>';
foreach($rows as $row) {
    echo '<tr>';
    foreach($columns as $column) {
        echo '<td>'.$row[$column].'</td>';
    }
    echo '</tr>';}
echo '</tbody></table>';?

## Scripting example:

*Compute HTML table from CSV file*

difficult to read and difficult to maintain
DEMO?

http://101companies.org/wiki/Contribution:php

- Show the implementation in action.
- Explain the architecture of the implementation.
- Show details of server-side scripting.
- Cover accidental complexity such as DB access.
Summary on SSS

You learned about ...

• the principles behind server-side scripting,
• the role of a Web server in Web applications,
• Apache Web server basic configuration,
• the generation of dynamic web pages with PHP,
• and, in fact, web programming with PHP.
Resources on SSS

• http://www.w3schools.com/php/default.asp
• http://www.php.net
REpresentational State Transfer (REST)
REST -- *a set of principles* that define how Web standards, such as HTTP and URIs, are supposed to be used in Web applications.
Resources and Resource Identifiers

- The key abstraction of information in REST is a resource.
- Each resource has a resource identifier.

Examples of identifiers

- http://example.com/customers/1234
- http://example.com/products/4554
- http://example.com/processes/salary-increase-234
Resources can have multiple representations, e.g., JSON/XML/HTML.
Example: the resource of ‘all companies’
Remember:
Hypertext Transfer Protocol


- GET Request representation for resource
- HEAD Like GET but without response body
- PUT Upload representation for resource
- POST Submit data for resource
- OPTIONS Query for available methods
- CONNECT Facilitate SSL-encrypted communication
- DELETE Delete specified resource
- TRACE Return request as it arrived at server
- PATCH Partial modification of resource
RESTful Web Service HTTP methods

- **Collection** URI, such as http://example.com/companies/

- **GET:** List the URIs and perhaps other details of the collection's members

- **PUT:** Replace the entire collection with another collection.

- **POST:** Create a new entry in the collection. The new entry's URL is assigned automatically and is usually returned by the operation.

- **DELETE:** Delete the entire collection.
RESTful Web Service HTTP methods

- **Element** URI, such as http://example.com/companies/32

- **GET**: **Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type.

- **PUT**: **Replace** the addressed member of the collection, or if it doesn't exist, **create** it.

- **POST**: Treat the addressed member as a collection in its own right and **create** a new entry in it.

- **DELETE**: **Delete** the addressed member of the collection.
DEMO

https://github.com/rlaemmel/startup14/

- Show the look and feel of the Polls app
- Explain the URL scheme in relation to REST
- Show JSON representation versus HTML
Properties of REST

• Client-server
• Stateless: user data is not stored between requests
• Cache

HTTP 304: Not modified
Summary

You learned about ...

• the REST architecture pattern,
• working with "resources" via HTTP,
• the JSON format,
• and some bits of Django.
Resources on REST

• A Brief Introduction to REST: http://www.infoq.com/articles/rest-introduction

Model-View-Controller (MVC) in Web Application Development
MVC - a classic definition

- The **Model** is the application object
- The **View** is its screen presentation
- The **Controller** defines the way the user interface reacts to user input
Model–View–Controller (MVC) is a computer software design pattern that separates the representation of information from the user's interaction with it. The model consists of application data and business rules, and the controller mediates input, converting it to commands for the model or view. A view can be any output representation of data, such as a chart or a diagram. Multiple views of the same data are possible.
The Model-View-Controller Architecture

here: server side

1. Browser sends request
2. Controller interacts with model
3. Controller invokes view
4. View renders next browser screen
The (Ruby on) Rails MVC

2. Routing finds Store controller
3. Controller interacts with model
4. Controller invokes view
5. View renders next browser screen

© 2014-2015, Ralf Lämmel, rlaemmel@acm.org
Model

- Maps to a table in a database. By convention, a model named Company will map to the database table companies, and the model will have a filename `company.rb` within `app/models` folder.

```ruby
class Company < ActiveRecord::Base
  validates :name, :presence => true
  has_many :departments
end

class Department < ActiveRecord::Base
  belongs_to :company
  belongs_to :department
  has_many :departments
  has_many :employees
end

class Employee < ActiveRecord::Base
  belongs_to :department
end
```
Controller

• Responds to external requests from the web server to the application, and responds to the external request by determining which view file to render

```ruby
class CompaniesController < ApplicationController
  # GET /companies
  # GET /companies.json
  def index
    @companies = Company.all

    respond_to do |format|
      format.html # index.html.erb
      format.json { render :json => @companies }
    end
  end
end
```

how to respond to certain HTTP requests
Controller (II)

- Handles *people-friendly URLs* extremely well.
- Manages *caching*, which can give applications orders-of-magnitude performance boosts.
- Manages *sessions*, giving users the impression of ongoing interaction with our applications.
View

• In the default configuration of Rails is an *erb* file. It is typically converted to output html at run-time.

```html
<div class="headline"><h2>101companies Ruby on Rails Web App</h2></div>
<div class="content">
<h1>Listing companies</h1>
<table>
  <tr>
    <th>Name</th>
  </tr>
  <% @companies.each do |company| %>
    <tr>
      <td><%= company.name %></td>
      <td><%= link_to 'Show', company %></td>
      <td><%= link_to 'Edit', edit_company_path(company) %></td>
      <td><%= link_to 'Destroy', company, :confirm => 'Are you sure?', :method => :delete %></td>
    </tr>
  <% end %>
</table>
<br />
<%= link_to 'New Company', new_company_path %>
</div>
```

Ruby embedded instead of PHP
Ruby

Ruby is a dynamic, reflective, general-purpose object-oriented programming language that combines syntax inspired by Perl with Smalltalk-like features. It was also influenced by Eiffel and Lisp.

Philosophy (Rails)

- **DRY** – “Don’t Repeat Yourself” – suggests that writing the same code over and over again is a bad thing.

- **Convention Over Configuration** – means that Django makes assumptions about what you want to do and how you’re going to do it, rather than requiring you to specify every little thing through endless configuration files.

- **REST is the best pattern** for web applications – organizing your application around resources and standard HTTP verbs is the fastest way to go.

Rails enforces the use of MVC as a pattern.
Example

101 implementation: rubyonrails

- Look into internals this time.
- Consider details of MVC in this implementation.
- Understand bits of Ruby on Rails philosophy.
Companies::Application.routes.draw do
  resources :employees

  resources :departments do
    resources :departments
    resources :employees
  end

  resources :companies do
    resources :departments
  end

  get "home/index"
end
rake routes

GET    /employees(.:format)                             {:action=>"index", :controller=>"employees"}
POST   /employees(.:format)                            {:action=>"create", :controller=>"employees"}
GET    /employees/new(.:format)                      {:action=>"new", :controller=>"employees"}
GET    /employees/:id/edit(.:format)                  {:action=>"edit", :controller=>"employees"}
GET    /employees/:id(.:format)                         {:action=>"show", :controller=>"employees"}
PUT    /employees/:id(.:format)                         {:action=>"update", :controller=>"employees"}
DELETE /employees/:id(.:format)                      {:action=>"destroy", :controller=>"employees"}
# GET /departments/1
# GET /departments/1.json

def show
  @department = Department.find(params[:id])

  respond_to do |format|
    format.html # show.html.erb
    format.json { render :json => @department }
  end
end
<div class="headline"><h2>101companies Ruby on Rails Web App</h2></div>

<div class="content">

<p id="notice"><%= notice %></p>

<div class="attr">

<p>
<b>Name:</b>

<%= @department.name %>
</p>

</div>

<hr>

<div class="attr">

<p>
<b>Manager:</b>

</p>

<% @department.employees.each do |employee| %>

<p>
<%= if (employee.isManager?)
    link_to employee.name, employee_path(employee)
  end %>
</p>

<% end %>

......
JavaScript and AJAX
JavaScript (sometimes abbreviated JS) is a prototype-based scripting language that is dynamic, weakly typed and has first-class functions. It is a multi-paradigm language, supporting object-oriented, imperative, and functional programming styles.


Standardized JavaScript = ECMAScript

http://www.ecma-international.org/ecma-262/5.1/Ecma-262.pdf
### TIOBE Index for June 2015

http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

<table>
<thead>
<tr>
<th>Jun 2015</th>
<th>Jun 2014</th>
<th>Change</th>
<th>Programming Language</th>
<th>Ratings</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>↑</td>
<td>Java</td>
<td>17.822%</td>
<td>+1.71%</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>↓</td>
<td>C</td>
<td>16.788%</td>
<td>+0.60%</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>↑</td>
<td>C++</td>
<td>7.756%</td>
<td>+1.33%</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>↑</td>
<td>C#</td>
<td>5.056%</td>
<td>+1.11%</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>↓</td>
<td>Objective-C</td>
<td>4.339%</td>
<td>-6.60%</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>↑</td>
<td>Python</td>
<td>3.999%</td>
<td>+1.29%</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>↑</td>
<td>Visual Basic .NET</td>
<td>3.168%</td>
<td>+1.25%</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>↓</td>
<td>PHP</td>
<td>2.868%</td>
<td>+0.02%</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td></td>
<td>JavaScript</td>
<td>2.295%</td>
<td>+0.30%</td>
</tr>
<tr>
<td>10</td>
<td>17</td>
<td>↑</td>
<td>Delphi/Object Pascal</td>
<td>1.869%</td>
<td>+1.04%</td>
</tr>
<tr>
<td>11</td>
<td>-</td>
<td>↑</td>
<td>Visual Basic</td>
<td>1.839%</td>
<td>+1.84%</td>
</tr>
</tbody>
</table>

JavaScript is a very important language :-)
Warning / Disclaimer

• JS might look very unnatural at first.
• It is essentially LISP with C-like syntax:
  ▶ Very powerful
  ▶ Very flexible
  ▶ Complicated due to some language design decisions

We will compare JS with Java, erratically.
A JavaScript joke

```javascript
[] + {}
"[object Object]"

{} + []
0
```
Basics of JS

Some ‘good’ part of JS:

```javascript
var add = function (a, b) {
    return a + b;
}
var y = add(2,3)
```

Define and apply a function.

Some ‘bad’ part of JS:

```
[] + {}
"[object Object]"
{} + []
0
```
The notion of prototype

```
Object.create = function (o) {
    var F = function () {};
    F.prototype = o;
}

meganalysis = {
    "name": "Meganalysis"
};
meganalysis2 = Object.create(meganalysis);
```

Everything is an Object, like in Smalltalk :-)
The Method Invocation Pattern

```javascript
var company = {
    total: 1000,
    increment: function(val) {
        this.total += val;
    }
};

company.increment(100);
console.log(company.total);
```

`this'/local scope: “company” object

company - object
total - public property
increment - public method

*Think in Java*: no classes???
The Function Invocation Pattern

```javascript
add = function (a,b) {
    console.log(this);
    return a+b;
}
x = add(2,3);
```

‘this’/local scope: a global object

JS runs in the web browser.
The global object is Window.
Constructor Invocation Pattern

// Create a constructor function for employees.
var Employee = function (name) {
    this.name = name;
};

// Give all employees a public method.
Employee.prototype.get_name = function () {
    return this.name;
};

// Make an instance of Employee.
var ralf = new Employee('Ralf');
name = ralf.get_name();
console.log(name);

Think in Java: constructor invocation
var **Employee** = function (name) {
    this.name = name;
};
**Employee**.prototype.get_name = function () {
    return this.name;
};
var ralf = **new Employee**("Ralf");
ralf.name = “Andrei”
name = ralf.get_name();

**Q:** What is the value of the name?

**A:** Think in Java: We need to ‘hide’ properties.
```javascript
var ralf = (function () {
    var name = "Ralf";
    return {
        getName: function () {
            return name;
        }
    }
})();
```

```javascript
ralf

```javascript
getName: function () { return name; }
```

```javascript
ralf.getName(): "Ralf"
```

“name” is hidden
var Person = function (name) {
    this.name = name;
    this.isHuman = true;
};
var Employee = function (name) {
    this.name = name;
};
Person.prototype.isHuman = function()
{
    return this.isHuman;
};
Person.prototype.toString = function()
{
    return '[Person "'+this.name+'"]';
};

// Here's where the inheritance occurs
Employee.prototype = new Person();

// Otherwise instances of Employee
// would have a constructor of Person
Employee.prototype.constructor = Employee;

Employee.prototype.toString = function()
{
    return '[Employee "'+this.name+'"]';
};
JS is not the ‘best’ OO language. Why should I care?

Because it’s the language in the Web browser:

Client-side scripting
Front-end development
Interactive web applications

} = JavaScript
HTML Document
Object Model

<html>
<head>
   <title>My title</title>
</head>
<body>
   <a href="#">My Link</a>
   <h1>My header</h1>
</body>
</html>
HTML DOM Event Handling

http://jsfiddle.net/DrGigabit/aQctY/1/
```html
<html>
  <head>
    <title>My title</title>
  </head>
  <body>
    <a href="#">My Link</a>
    <h1>My header</h1>
    <button id="createButton">Click me</button>
  </body>
</html>

var button = document.getElementById("createButton");
button.addEventListener("click", function() {
  alert("Click!");
}, false);

asynchronously = interactive UI
```
Function as arguments (callbacks)

// Define a function on two number args and a function arg.
function randomBlock(arg1, arg2, callback) {
    // Generate a random number between arg1 and arg2.
    var rnd = Math.ceil(Math.random() * (arg2 - arg1) + arg1);
    // Pass the result to the function argument.
    callback(rnd);
}

// Apply randomBlock to an anonymous function.
randomBlock(5, 15, function(arg) {
    // This anonymous function will be applied later.
    console.log("Callback called with arg = " + arg);
});
Motivating scenario: Asynchronous input/output

Make a request *synchronously*

request = prepare_the_request(...);
response = send_request_synchronously(request);
zzzzZZZZZZZZzzz <--- Waiting time
display(response);

Make a request *asynchronously*

request = prepare_the_request(...);
send_request_asynchronously(request, function (response) {
    display(response);       <--- When ready
});
doSomethingElse();
jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development.
jQuery

```javascript
var button = $('#createButton');
button.click(function(){
  alert('clicked');
});

plain JS

```javascript
var button = document.getElementById("createButton");
button.addEventListener("click", function() {
  alert("Click!");
}, false);

```

```javascript
$('#createButton') === document.getElementById("createButton");
```
Another DOM Manipulation

h2>Greetings</h2>
<div class="container">
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>

+<p>Test</p>;

=  

<h2>Greetings</h2>
<div class="container">
  <div class="inner">
    Hello
    <p>Test</p>
  </div>
  <div class="inner">
    Goodbye
    <p>Test</p>
  </div>
</div>
Asynchronous JavaScript and XML (AJAX)

Motivation
We know how to do client-side programming in JavaScript. How do we interact with the server?
What’s AJAX?

• AJAX = Asynchronous JavaScript and XML
• Make asynchronous requests to the server.
• Receive response eventually through callback.
• Support based on XMLHttpRequest object.
• ‘No page refresh’
AJAX example: loading company data from the server

```javascript
var company = {};  
company.response;  
company.loadData = function() {  
  var xhr = new XMLHttpRequest();
  xhr.open('GET', 'company.xml', true);

  xhr.onload = function(e) {
    if (this.status == 200) {
      company.response = xhr.responseXML;
      controller.loadInner();
    }
  };

  xhr.send();
}
```

- **Prepare request object**
- **Point to resource**
- **Register response handler**
- **Send actual request**
DEMO

101implementation:html5XMLHttpRequest

Show XHR (XmlHttpRequest) in a 101implementation.
Cascading Style Sheets (CSS)
What is CSS?

• Cascading Style Sheets (CSS) is a language for specifying how documents are presented to users.

• A document is a collection of information that is structured using a markup language.
Without CSS

<head>
<title>CSS Example</title>
</head>
<body>
<H1>Very important</H1>
42
<H2>Less important</H2>
88
</body>
With CSS

<head>
<title>CSS Example</title>
<style type="text/css">  
 H1 { font-size: x-large; color: red } 
 H2 { font-size: large; color: blue } 
</style>
</head>
<body>
<H1>Very important</H1>
42
<H2>Less important</H2>
88
</body>

Very important
42
Less important
88
Why do we need CSS?

- Provides a powerful and flexible way to control the details of displaying web documents.
- HTML is more concerned about the content; CSS is used to impose a particular style on the document.
- Named ‘cascading style sheets’ because they can be defined at three different levels to specify the style of a document:
  - Inline (per element), document level, external.
Vary CSS for the shown HTML markup.

http://jsfiddle.net/rlaemmel/eVbY7/
Format of style rules

Remember HTML DOM?

selector { property1: value1; property2: value2 }

H1 { font-size: x-large; color: red }
Rules with an \textit{id} selector

CSS

\texttt{name \{ text-indent: 3em \}}

HTML

\texttt{<p id="name">Andrei</p>}

Specify a style for a single, unique element.
Rules with an *class* selector

CSS

```css
.center {text-align: center;}
```

HTML

```html
<p class="center">Some text</p>
```

Specify a class of style which can be choosen.
Rules with ‘class’ in HTML5

http://jsfiddle.net/rlaemmel/MgPxj/3/
Layout in HTML5

http://jsfiddle.net/DrGigabit/ZbV3F/2/
The End