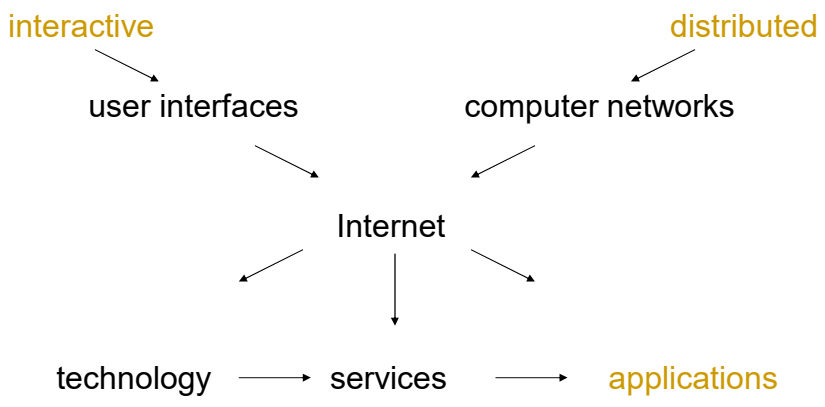
Interactive Distributed Applications

Prof. Dr. Tom Rüdebusch
Communication and Media Engineering

Hochschule Offenburg
University of Applied Sciences

Course Topics



5 ECTS Credits are 150 Hours of Work

NO literature reviews
NO papers to turn in

BUT follow-up work after each class

- revisit all examples, redo all demos on your computer
- recap your learning outcomes
- explain them to your fellow students
- prepare questions for next class

AND implement all exercises on your computer

Objectives

- to recognize the Internet as a world-wide network infrastructure and to be able to explain, evaluate and exploit its services
- to appreciate the World Wide Web as an interactive distributed system and to be familiar with Web technologies
- to recognize applications as the decisive factor within the Internet and to think about new applications
- to set the context for self-guided learning on demand

Course Organization

- slides + notes + examples + demonstrations
- slide copies, examples, messages in Moodle
- computer-based exercises
- written exam (90 min, cheat sheet provided)

! These are just slides. It is very important to take notes for all explanations and demonstrations during the lectures.

Web Links

- searchable RFC archive
  - https://datatracker.ietf.org/
- Internet well-known ports
  - https://www.iana.org/assignments/port-numbers
- Vannevar Bush: As We May Think. In: The Atlantic Monthly, July 1945
  - https://www.w3.org/History/1945/vbush/vbush-all.shtml

Web Links (2)

- World Wide Web Consortium (W3C)
  - https://www.w3.org/
  - local search: " HTML5", "CSS", "XHTML 1.0 DTD"
- markup validation service of the W3C
  - https://validator.w3.org/
  - https://jigsaw.w3.org/css-validator/
- JavaScript, HTML, CSS, HTTP for Web developers
  - https://developer.mozilla.org/docs/Web

Web Links (3)

- browser compatibility
  - https://caniuse.com/
- PHP reference
  - https://php.net/
- collection of XMLapplications
  - http://xml.coverpages.org/siteIndex.html#toc-applications

Web Links (4)

- XML Copy Editor
  - http://xml-copy-editor.sourceforge.net/
- syntax check of DTDs
  - https://www.xmlvalidation.com

Contents

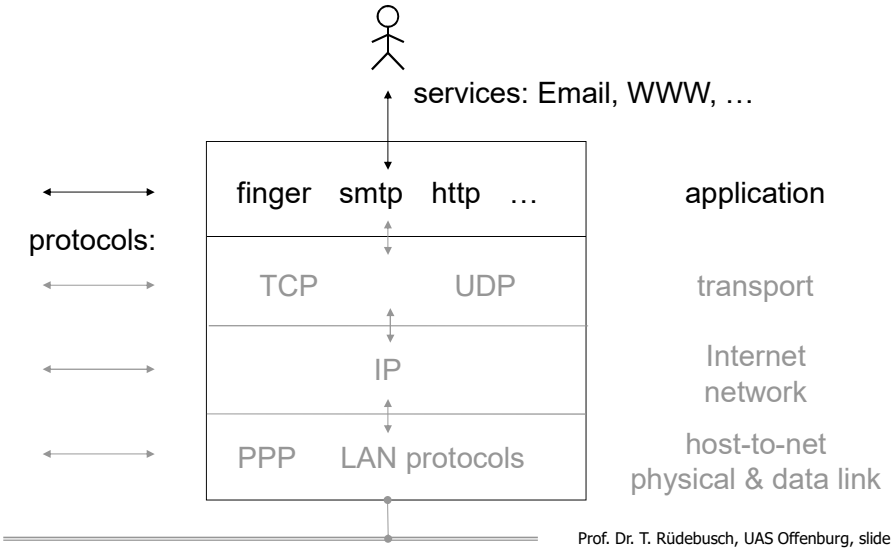
Internet Services

The building blocks of the application

Objectives

- to recognize Internet services as the first interface to the end user
- to understand the basic architecture of Internet services

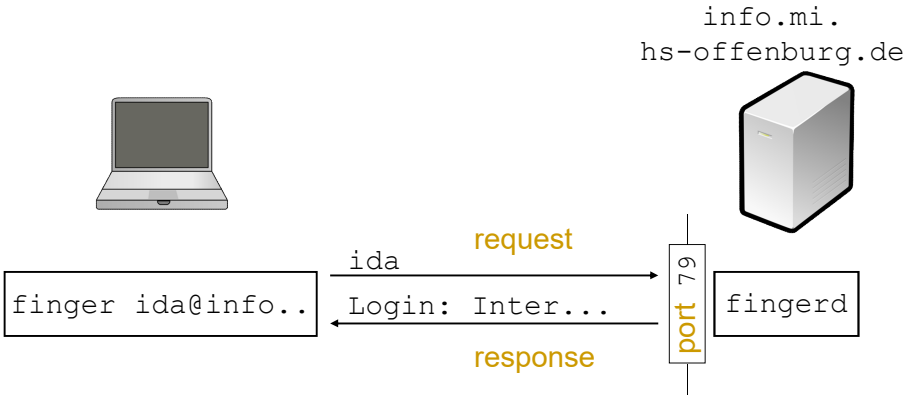
Internet Layers



Protocol Standards

- a defined set of rules and formats for computer communication is called a protocol
- the functionality of each layer is implemented employing the respective protocol and provided as a service to upper layers
- without international standards for protocols, world-wide communication would not be possible (Internet: "Request for Comments", RFC)

Client-Server Architecture of Internet Services



finger client | finger protocol | finger server

Telnet (SSH)

`telnet` *`host`*

- interactive, text-based login to remote computers ("remote login")
  - network connection from Telnet client (local host) to Telnet server (remote host)
  - local input is transmitted to remote host
  - remote output is transmitted to local host
- Unix command line ("shell")
  - `man` *`command`*, `ls -l`, `echo`, `cat`, `more`, `mkdir`, `cd`, `pwd`, `mv`, `cp`, `rm`, `logout`
  - `~` `..` `.`
  - stdin `<` stdout `>` `>>` pipe(line)s `|`

Exercise "Telnet (SSH)"

- From your PC, login to host `info.mi.hs-offenburg.de` as user `ida`.

  Create a text file (make up an unambiguous file name from your personal name). Display the file and log out.

Telnet (2)

`telnet` *`host port`*

- "generic client"
  - Internet protocols within the application layer employ character streams!
  - use Telnet as a character-transmitting connection

Exercise "Telnet as a Generic Client"

- Retrieve information on users currently logged in at `info.mi`, using the finger client on a (Windows) command line.
- Do the same thing by executing the finger protocol using generic Telnet.
- Some Unix computers offer a service called "daytime" that responds with the current local time. The daytime protocol defines that a connect to the daytime server is a daytime request.
  Your PC does not implement a daytime client. Nevertheless, determine the local time within another country.

FTP (SFTP)

- (command line: `ftp` *`host`*)
- GUI FTP clients

- file transfer between local and remote computers (File Transfer Protocol)
  - "upload" to, "download" from remote host

Exercise "FTP"

- Download the file that you created during the Telnet exercise to your local PC.

- Write a new text file on your local PC, upload it to `info.mi` and display it there on the command line.

Intermediate Test "Internet Services"

- Show the architecture of the Internet service WWW in a diagram (use `developer.mozilla.org` in your drawing). Make sure to use all important terms in their specific form for the WWW.

- Which parts of your diagram belong to the application layer, which part is relevant to the transport layer, which two parts are relevant to the Internet layer?

Objectives Revisited

- to recognize Internet services as the first interface to the end user
- to understand the basic architecture of Internet services

The World Wide Web

Interactive and distributed

Objectives

- to recognize the World Wide Web as an interactive distributed system drawing on both areas, user interfaces and computer networks
- to classify the WWW as an Internet service
- to know how the Web works
- to understand the various Web technologies and to be able to employ them in an application-driven way

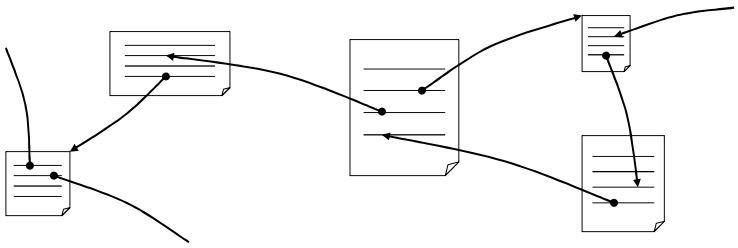
The WWW is...

- the Internet "killer application (service)"
- another evidence for the UI's importance
- intuitive through point-and-click
- a hyper structure through HTML
- two-way through forms
- another Internet service
- a client-server system using HTTP above TCP/IP

...and...

- open
  - to different platforms
    - open standards (TCP/IP, HTTP, HTML,...)
  - to any multimedia format
    - MIME ("Multipurpose Internet Mail Extensions"), plug-ins
  - to external IT systems
    - client: plug-ins, JavaScript, ...
    - server: CGI, PHP, Node.js, ...

Hypertext



- "non-linear text", associative structure
- information entities (pages, nodes)
- links (hyperlinks)
  - source
  - destination

URL

- Uniform Resource Locator
- more general: Uniform Resource Identifier URI

- unique address of/within a information entity
- link in HTML
  - `<a href="destination">source</a>`
- spanning host boundaries by specifying the destination host

URL (2)

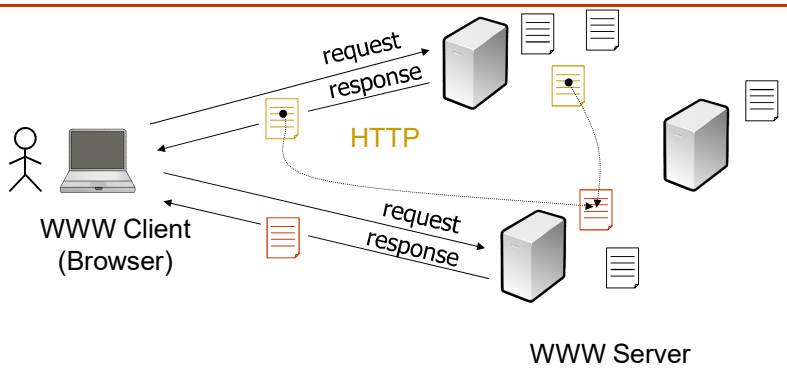
- basic structure

  protocol`://`*destination_host*`/`*description*

  - `https://info.mi.hs-offenburg.de/~tom/Misc/index.html`
  - `info.mi.hs-offenburg.de/~tom/Misc/`
  - `file:///C|/WINDOWS/win.ini`

WWW System

Protocol



- retrieve a (multimedia) information item:
  Hypertext Transfer Protocol

After Clicking on...

"Hello ..." in the yellow Web page

1. the browser determines the URL `https://info.mi.hs-offenburg.de/~tom/red.html` from the page source
2. the browser opens a connection to port `80` at host `info.mi.hs-offenburg.de`
3. the browser sends the HTTP request

```
GET /~tom/red.html HTTP/1.1
Host: info.mi.hs-offenburg.de
empty line
```


After Clicking on... (2)

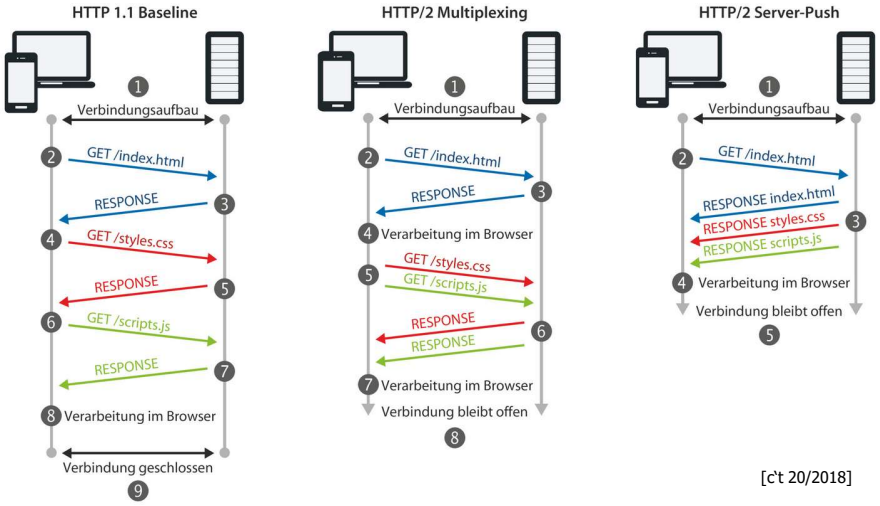
4. the server sends the file `red.html` within its HTTP response
5. the server closes the TCP connection (after time-out)
6. the browser displays the text of `red.html`
7. the browser subsequently retrieves all the images,... within `red.html` in the same way and displays them

- Web servers are very picky about correct requests. Even a redundant space at the end of a line will result in an error message. Keep this in mind when you try this demo for yourself.

HTTP(S)

- is a stateless protocol
  - transport layer connect,
    HTTP request, response, …,
    transport layer disconnect
- HTTPS employs TLS between application layer and transport layer
- transfers data (Web pages, images, …) of any type
  - type named within MIME header
  - HTTP supports binary transfer

HTTP/2



[c't 20/2018]

HTTP/2 vs HTTP/3

- HTTP/2 - transport layer TCP
- HTTP/3 transport layer QUIC (Quick UDP Internet Connections)
  - connection-oriented over UDP
  - multiplexing on the transport layer
  - integrates TLS

HTTP Request

- request line (REST APIs)
  `GET, POST, PUT, DELETE, ...`
- request header
  `Host:, User-Agent:, Cookie:, ...`
- *empty line*
- entity body
  e.g. form entries with `POST`

HTTP Response

- status line
  status code (`2XX: Success, 3XX: Redirection, 4XX: Client Error, 5XX: Server Error`)
- response header
  `Content-Type:, Last-Modified:, Refresh:, Location:, Set-Cookie:, ...`
- *`empty line`*
- entity body
  object (Web page, file)

Exercise "HTTP"

- Retrieve the Web page with the URL

  `https://info.mi.hs-offenburg.de/tom/Misc/index.html`

  by executing HTTP over a generic telnet connection.
  Note: For this purpose, it is necessary to interpret the HTTP response carefully. Store the page in a file and display it within a Web browser (after removing the HTTP parts).

- Now, retrieve the same page with Firefox, having the developer tools open, and cache deactivated checked. What do you observe?

HTML

Page Description

- Hypertext Markup Language

- is the page description language of the Web
- is not a programming language

- describes Web pages in textual form
  - HTML: `these are <em>three important words</em> !`
  - browser: these are *three important words* !

Logical Structure and Layout Rules …

1. logical structure of a document
   - chapter, heading, paragraph, enumeration, figure, emphasis, …
2. is translated into
   - style (layout rules)
3. layout structure (physical representation)
   - (page,) text block, line, font, alignment, color, …

⇒ separation of logical document description and style description

… in the World Wide Web

1. logical structure of a document and
   - HTML source
2. styles
   - Cascading Style Sheets
   - (browser-specific layout rules)
3. are interpreted by the Web browser for presentation

- HTML versions: 1.0, 2.0, 3.2, 4.01, 5, XHTML 1.0, 1.1

Tags

- tags (markups) define elements within a Web page and are enclosed in pointed brackets
  - start tag (< … >) and end tag (</ … >)
  - lower case
  - with named parameters (attributes) and assignment of values with ="…"
  - properly nested, e.g. basic structure of a Web page:

Structure of HTML Sources

- identification as an HTML document

  `<html lang="en">`

  - encloses complete source

- header section

  `<head> … </head>`

  - instructions related to the whole document

- main body

  `<body> … </body>`

  `</html>`

Basic Tags

- titel

  `<title> … </title>`

  - related to the whole page, part of the header
  - appears in window title bar and within bookmarks

- headings

  `<h1> … </h1>,… <h6>...</h6>`

  - 6 levels of hierarchy

- paragraphs

  `<p> … </p>`

Basic Tags (2)

- ordered list
  `<ol> … </ol>`
  - type of numbers with CSS
- unordered list
  `<ul> … </ul>`
  - bullet symbol with CSS
- list item
  `<li> … </li>`
  - also mandatory for a nested list

Core Attributes

- can be used with any tag
  - unique identifier of a element
    `id="…"`
  - name of an element
    `title="…"`

Logical Styles

- describe the enclosed text element logically
- physical presentation is realized within the browser/CSS

```
<em> … </em>
<strong> … </strong>
<samp> … </samp>
<dfn> … </dfn>
```


Special Characters

- "character entities"
  - named
    `Ä ä ß à …`
  - numerical (ISO-Latin-1/Unicode)
    `Ä ä …`
  - reserved characters
    `< > & "   € …`

Declaring a Web Page

- correct document type
  `<!DOCTYPE html>`
- character set (within header section)
  `<meta charset="UTF-8">`
- comments
  `<!-- author, date, … -->`
  - may span several lines

page1.html

Page Formatting

- line break
  `<br>`
- horizontal ruler
  `<hr>`
- preformatted text
  `<pre> … </pre>`
- grouping of elements
  `<div> … </div>`
- text-level spans
  `<span> … </span>`

Links

- Web
  - = information entities + links
  - = "Hypertext"

- link = source + destination

  `<a> … </a>`
  - encloses anchor (text, image, …) being the link source
  - attribute `href="destination_URL"` defines link destination

Local Link Destination

- absolute path name
  - access path through complete folder hierarchy, always starts with `/`
  - e.g. `href="/assets/images/logo.gif"`

- relative path name
  - relative to the location of the HTML document containing the link
  - e.g. `href="info.html"` (`hires/company.gif`, `../../main/main.html`)
  - portable!

Remote Link Destination

- complete URL (Uniform Resource Locator)
  `protocol://host_name/path`
  - diverse Internet services (protocols)
  - any document type (file extensions)
  - e.g. `href="https://www.hs-offenburg.de/mi/mi.html"`
  - special characters within URLs
    - with `%ASCIICode` e.g. `%20` ( ), `%3f` (?), `%2f` (/)
    - e.g. `href="https://my.sys.de/bad%20name%3f.html"`

Link Destination

- is an element within a page
  - using the id of the element after #
  - e.g. `href="mi.html#People"`
- opens in a new window
  - attribute `target="window_name"` for a named and reused browser window
  - always a new window with attribute value `_blank`

page2.html

Images

- images
  `<img>`
  - attributes
    `src="image_URL" alt="description"`
    `height="pixels" width="pixels"`
- as link sources
  - "image links", "clickable images"
  - nesting image tag within anchor tag
- external images
  - new page for link destination

page3.html

Multimedia Contents and Semantic Elements

- embedding multimedia content
  `<iframe>, <canvas>, <video>, <audio>`
- embedding content in dedicated markup
  `<svg>, <math>`
- semantic elements
  `<header>, <footer>, <section>, <article>, <nav>, …`

Creating HTML Documents

1.

2.

3.

4.

Checking HTML Source Code

- W3C Validator
- Firefox Web Developer

Exercise "HTML"

- Create a few linked HTML pages using your favorite text editor. Make use of the tags and attributes that have been discussed so far.
- Display the pages locally within a Web browser.
- Validate your pages at `https://validator.w3.org`
- Examine the page structure using the Web Developer tools.
- Transfer the pages into a personal subdirectory of `public_html`, account `ida` on `info.mi` using a graphical FTP client, and retrieve them with your Web browser.

Forms

- user feedback

  `<form> … </form>`
  - attributes `method="post" (get)`
    `action="script_URL"`

- <u>UI elements</u>

  `<input>`
  - attributes `type="text" (password, radio, checkbox, hidden, submit, reset, button, file, number, email, date, …)`
    `name="element_name" value="value"`
    `checked placeholder required min max …`

Forms (2)

- structuring forms

  `<label> … </label>`

  - associates a UI element with text (including markup)

  ```
  <fieldset>
    <legend> … </legend>
     …
  </fieldset>
  ```

  - groups UI elements and other HTML elements
  - labels the group

form.html

Exercise "Forms"

- Develop an order form for a product of your choice. Make use of the UI elements on the previous slides. Use the script `/~tom/formGetPost.php` on the host `info.mi` as the `action` value.
- Display the pages locally within a Web browser.
- Validate your pages at `https://validator.w3.org`
- Examine the page structure using the Web Developer tools.
- Transfer the pages into a personal subdirectory of `public_html`, account `ida` on `info.mi` using a graphical FTP client, and retrieve them with your Web browser.

Static/Dynamic Web Resources

Server

referring to the time of the request to the Web server

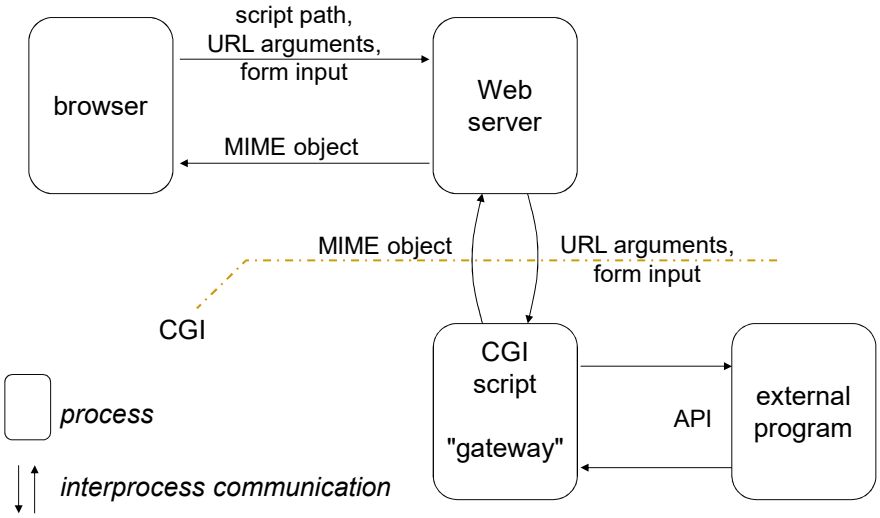
- static
  - resource exists as a file
  - e.g. image, style sheet, JavaScript framework, …
- dynamic
  - resource is created by a server-side program
  - e.g. HTML source code, image, …

Common Gateway Interface (CGI)

- between Web server and CGI script to dynamically create MIME objects
- advantages
  - supported by all Web servers
  - to be used with any programming language
  - CGI script implements "gateway" to external software
  - RFC3875
  - fast when using FastCGI

  ! true understanding of dynamic Web applications

CGI Execution



Employing CGI Scripts from HTML Code

- script URLs
  - `<a href="script URL">`
  - `<form action="script URL">`
  - `<img src="script URL">`

- recognized on the Web server by folder name (e.g. `cgi-bin/`) or extension (e.g. `.php`)

Implementing CGI Scripts

1. write source code (e.g. on a PC) in a programming language of your choice,
2. transfer it to the Web server,
3. compile it on the server,
4. identify executable as CGI script (folder or extension, resp.), and
5. grant execution rights for the user associated with the Web server process

a) CGI – Output

to the Web server via standard out

- as a MIME object
  - `Content-Type: text/plain, text/html, image/gif,…`
  - *empty line*
  - entity body

Examples "CGI Output"

- a dynamically created HTML page to display the current date and time

  date.html

Exercise "CGI Output"

- Implement a CGI script in C that displays all users currently logged in on the Web server.
  Hint: Make use of the shell command "who" and the HTML `<pre>` element to keep the output in columns.

  Make your own subdirectory in `public_html/cgi-bin/`!

- Call this script from the address line of your browser.

b) CGI – Environment Variables

from the Web server via global variables:

- client
  - `HTTP_USER_AGENT, REMOTE_ADDR, HTTP_COOKIE, …`
- request
  - `REQUEST_METHOD, QUERY_STRING, CONTENT_LENGTH, …`
- server
  - `SERVER_SOFTWARE, SERVER_NAME, …`

c) CGI – Input via URL Arguments

- URL arguments are located after ?
  - e.g. `https://info.mi.hs-offenburg.de/tom/cgi-bin/getmydate?IDA`
- server-side environment variable `QUERY_STRING`
  - e.g. `QUERY_STRING=IDA`

date.html

Exercise "CGI Input via URL Arguments"

- Write down the source code for a CGI script `url_argument.c` that is called by the URL

  `https://info.mi.hs-offenburg.de/~ida/cgi-bin/`*`yourSubdirectory`*`/url_argument?`*`number`*

  and creates an HTML heading level *number* with the text content "I am a heading level *number*". (*number* is a placeholder for an integer between one and six.)

d) CGI – Input via Form Fields

- encoding of form input by the browser as name/value pairs:
  name1=value1&name2=value2&…
  e.g. `theName=Tom+R&thePasswd=&theSize=small`

- transferred to the Web server with `method="get"` as URL argument or
- transferred to the Web server with `method="post"` as entity body of the HTTP request from the Web server to the script via standard in (`CONTENT_LENGTH` specifies number of characters)

Intermediate Test "WWW System"

TOC

- On the Web server `sell.stuff.net`, in the `public_html` directory of the user `salesman`, there is an HTML form with the following lines

```
<form action="cgi-bin/justDoIt">
<input type="checkbox" name="agree" value="yes">
```

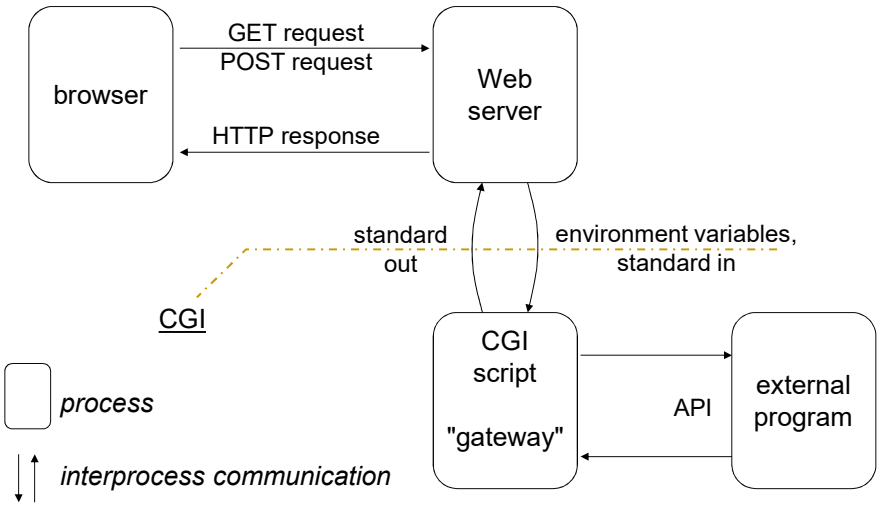
- How can you use telnet in the console of your computer to make the Web server believe that this form was sent with the checkbox confirmed?
- What is the full URL you need to enter in the address line of a browser to achieve the same?

Example "CGI Input via Form Fields"

- asterisks pattern in a dynamic Web page

  asterisks.html

CGI Execution Technically



Intermediate Test "CGI"

- Write down the source code `superwho.c` for a CGI script that creates a heading "Hello" followed by a paragraph with the text "I am " and two links "Superman" and "Superwoman". The links call the same script again, this time matching the heading to the link clicked: "Hello Superman" and "Hello Superwoman" respectively.

- The generated web page must validate according to HTML5 without errors.

CGI Alternatives

- disadvantages of CGI in C
  - static parts of Web pages have to be created by program statements
  - no HTML editors can be used

⇒ "server-parsed HTML"
  - static HTML template + dynamically computed parts

PHP

- PHP: Hypertext Preprocessor

  `<?php … ?>`

- C/Java syntax, but
  - `$variablenname`, without data type
  - `echo` to standard out (a)
  - `header()`
  - `.` for string concatenation

date.html

Form Processing in PHP

- environment variables (b) and URL arguments (c) and form fields (d) in associative arrays
  - `$_SERVER ['QUERY_STRING']`
  - `$_GET['theName']`
  - `$_POST['theName']`
  - `isset(), empty()`

pizza.html

Exercise "PHP"

- For your order form from the exercise "Forms", write a PHP script that displays an order confirmation in the browser. Store this PHP script in the same folder as your order form.

  Notice: If you implement a POST form, the URL of the action attribute must contain the tilde ~ character, because a redirect (cf. the exercise "HTTP") always leads to a GET request.

- Check that the generated order confirmation validates according to HTML5.

Passing State Information

- in HTTP via
  - URL arguments — stateURL.php !
  - form fields — stateForm.php ! stateFormHidden.php
  - cookies — stateCookie.php
  - PHP manages `$_SESSION[]` and associated session id automatically, using URL arguments or cookies

Intermediate Test "PHP – Form Fields"

- Write a PHP script `login.php` that creates a form with a heading and a text input field. When the script is first called (`isset()`), the heading reads "What is your name?" When a name is then entered, the form reappears and the heading reads "Hello *Name*." If the form is submitted without entering a name (`empty()`), the heading on top of the form will read "Hello Nobody!"
- The name entered must not be visible in the browser address line.
- The generated web page must validate according to HTML5 without errors.

Intermediate Test "PHP – URL Arguments"

- Implement the intermediate test "CGI" in PHP.
- The generated web page must validate according to HTML5 without errors.

Cascading Style Sheets (CSS)

Client

- layout rules ("style sheet")
  - translation of logical document structure (HTML source) into physical presentation
- Cascading Style Sheets
  - often, several layout rules are applicable (e.g. as a single HTML element, as an element type, as a nested element etc.)
  - "cascade" defines precedence
- objectives
  - separation of logical structure and layout
  - comprehensive control over the page design

Layout Rule

```
selector {
   style_property1: value1; /*declaration1*/
   style_property2: value2; /*declaration2*/
   …
}
```

- the selector determines the HTML elements that the rule applies to
- one or more declarations determine the physical representation of all elements that match the selector

Style Properties – Font

font-family

- `times, helvetica, garamond, …`
- generic names: `serif, sans-serif, monospace, …`
- list alternatives, e.g. `{font-family: garamond, times, serif}`

font-size

- units: `em, px, pt, in, cm, %, …`

font-style

- `normal, italic, …`

font-weight

- `100,…, 400(normal),…, 700(bold),…, 900`

Style Properties – Text

text-decoration

- `none, underline, line-through, …`

line-height

- same units as `font-size`

text-align

- `left, center, right, justify`

Further Style Properties

color

- `black, white, orange, …`
- `rgb(0,0,0), rgb(255,255,255), …`

background-color

- colors as above or `transparent`

float

- `none, left, right`

display

- `block, inline, inline-block, none, list-item, …`

Using a Style Sheet

- save as a style file with the extension `.css`
- connect to HTML file in its page header

```
<link rel="stylesheet" href="style_URL">
```

- one style sheet for all the Web pages of an organisation
- multiple style sheets for one Web page

style1.css

Selectors

- element type

  HTML_element_type

- class

  HTML_element_type.class_name

  - with HTML attribute `class="class_name"`

- (one unique) element

  #identifier

  - with HTML attribute `id="identifier"`

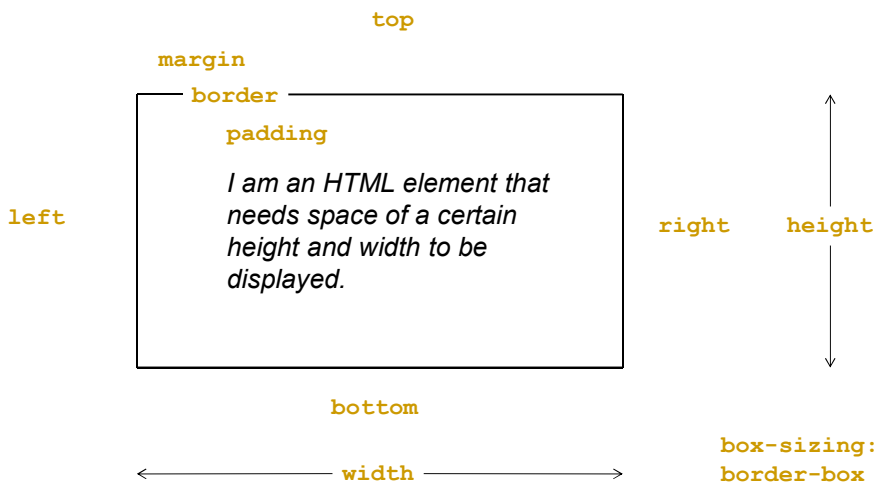
Selectors (2)

- attribute
  `[attribute="value"]`
- combinators
  - descendants *space*, …
- grouping
  selector1, selector2, …
  - possibly adding/overwriting style declarations for single selectors

Selectors (3)

- pseudo-classes (state)
  - `:hover, :focus, :first-child, :required, :invalid, :checked, …`
- pseudo-elements (element part)
  - `::first-letter, ::placeholder, …`

CSS Box Model



Style Properties – Positioning

position

- relative to the location without positioning: `relative`
- in coordinates of a positioned parent element (or else in `body` coordinates): `absolute`
- "non scrolling region", always positioned absolutely: `fixed`

left, top, right, bottom page4.html

- offset from left, top, right, bottom
- units: `em, pt, px, in, cm, %, …`

z-index

- location of a positioned element within a stack of overlapping positioned elements
- `1, 2, 3, …`

Exercise "CSS"

- Design an attractive layout for your order form as an external style sheet.
- Validate your style sheet at
  `https://jigsaw.w3.org/css-validator/`
- Now use this layout also for the order confirmation by adding the link to the external style sheet to your PHP script.

Choice of Styles

- cascade (per property!)
  - inline above embedded above external style sheet
  - specificity (id above class above element type)
  - sequence within the CSS source code
  - and: nesting of HTML elements
- media queries
  `@media` *`query`* `{` *`Style Sheet`* `}`
  - media types: `all, screen, print`
  - media features: `(max-width: 400px), (orientation: landscape), …`

Intermediate Test "CSS"

- Write a style sheet (and an HTML page to test it) for the following layout
  - If a class "important" is applied to an HTML element, its text content is displayed in double font size.
  - Emphases in paragraphs move down 2 pixels while the mouse pointer hovers over them.
  - The text in text input fields is blue.

- The Web page must validate to HTML5 without errors, and the style file must validate to CSS3.

JavaScript

- <u>object-oriented scripting language</u>
  - ECMAScript 2015…2024: `'use strict';`

- different from Java (similar syntax)
  - prototype-based inheritance, weak typing, interpreted (JIT compiler)
  - "Mocha", "LiveScript" (NS), TypeScript (Microsoft)

- browser (client)
  - interactivity of Web pages
  - user input validation before server call
  - source code in browser

JavaScript in the Browser

- HTML
  - content
  - logical structure

```
<p>
```

- CSS
  - presentation
  - layout

```
p {text-align: center;}
```

- JavaScript
  - behavior
  - interactivity

```
p.addEventListener('click',
      JavaScript);
```


Programming in JavaScript

- client and server ("Full Stack")
  - MEAN stack (MongoDB, Express.js, Angular, Node.js)
  - MERN stack (MongoDB, Express.js, React, Node.js)
- mobile apps
  - native, hybrid, Web app
- desktop applications
  - cross-platform (Electron)

Object Orientation

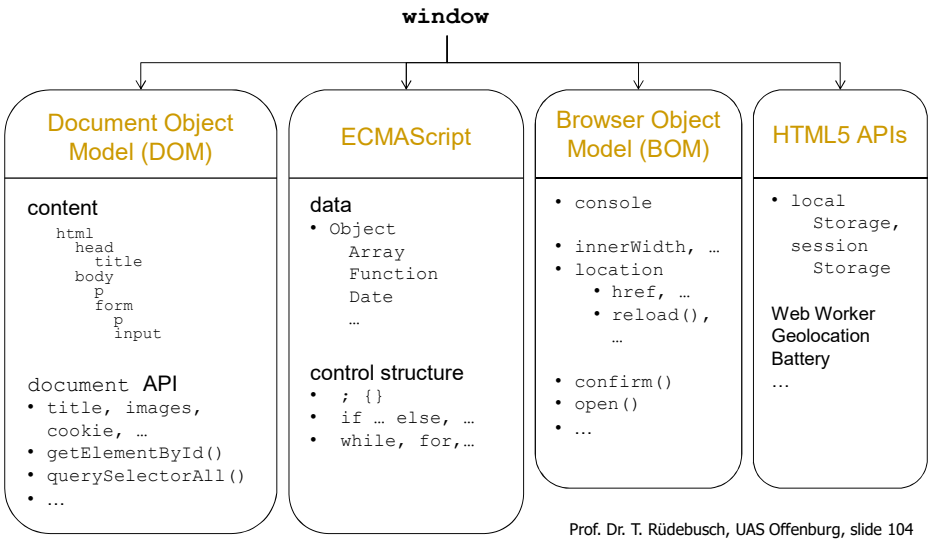
- properties (instance variables, object state)

```
objectName.variableName
    document.title
    location.href
    window.innerWidth
```

- methods (instance method, object behavior)

```
objectName.methodName(arguments)
    document.getElementById()
    location.reload()
    window.open("URL")
```


Object Orientation in the Browser



Exercise "Object Orientation in the Browser"

- Draw the complete document tree of the Web page

  `https://info.mi.hs-offenburg.de/~tom/stateForm.php`

- Open your order form in your browser. Try out in the Web Console:
  - the instance variables `title, images, innerWidth, location.href`
  - the functions `getElementById(), querySelectorAll(), confirm()`

JavaScript Embedded in HTML Source

- in the `head`

  `<script> … </script>`
  - Attribute
    `src="JavaScript-URL", defer`

- in the `body`

```
...
<script src="https://framework.com/fw.js"></script>
<script src="my.js"></script>
</body>
```

Data Types and Variables

- dynamic, weak typing
  - primitive types ("immutable")
    `number, boolean, string, …`
  - object/reference types `Object`
    `Array, Date, Function, …`
  - `typeof, instanceof`

- static scopes, blocks (with `let`)

```
let variableName = value;
   let input = "ida";
   input = 10;
   input = new Date();
```


Strings and Arrays

- strings

```
let answer = "Yes" + ' or ' + "No";
answer.length, …
answer.toLowerCase(), …
```

- arrays

```
let coord = new Array(10);
let people = ['Chick', 'Ella', 'Count'];
coord.length, …
people.forEach(), …
   for (i=0; i<people.length; i++)
      console.log('Name: ' + people[i]);
```

Operators and Control Structures

- operators

```
+ - * / % ++ -- += … === !== == != <= …
|| && …
```

- sequence

```
; {}
```

- choice

```
if … else, switch … case
```

- iteration

```
while, do … while, for
```


Functions (1)

- defining a function

```
function functionName(parameters) {
  variable definitions
  algorithm
  return functionValue;
}
```

- e.g.

```
function printMe(item) {
   console.log('Name: ' + item);
}
```

Functions (2)

- … are objects
  `typeof printMe, printMe instanceof object`
- … can be passed as arguments to other functions
  `people.forEach(printMe), coord.forEach(printMe)`
- … without a function name are called anonymous functions
  ```
  people.forEach(
   function (item) {console.log('Name: ' + item);} )
  ```
- arrow functions are compact anonymous functions
  `people.forEach(item => console.log('Name: '+item))`

Event-Based Programming

- defining event listeners
  `element.addEventListener('event', function);`
  always:
  `document.addEventListener('DOMContentLoaded', init);`
- passing event object ("e")
  `.target, .target.value, .target.textContent, …`
  `.preventDefault(), …`
- events
  `click, focus, blur, mouseover, mouseout, input, change, submit, reset, …`

events.html

Exercise "Event-Based Programming"

- A Web page consists of two buttons, one is inscribed with "click me", the other has no inscription. If the "click me" button is clicked, the inscription disappears and the other button is inscribed with "click me". This always alternates.
- Implement this behavior in HTML and JavaScript.
- It doesn't work?
  - Validate your Web page.
  - Do you see syntax or runtime errors in the console when loading or executing?
  - Print test messages using `console.log()`.
  - Set breakpoints in the debugger!

Document Object Model (DOM)

- content
  - document tree, root is `document`
  - children are element nodes, attribute nodes, text nodes
- API (instance variables, instance methods)
  - selecting

    `document.getElementById('`*HTML id*`'),`
    `document.querySelectorAll('`*CSS selector*`'), …`
    element`.parentElement, .children, …`
  - modifying

    element`.textContent, .className, …`
  - adding

    `document.createElement(), `*element*`.append(), …`
  - removing

    element`.remove()`

Architectures of Web Applications

- Client-Side Rendering (CSR) — toDoList.html
  → Single Page Application (SPA)
  → Progressive Web App (PWA)

- Server-Side Rendering (SSR) — toDoList.php

- Static Site Generation (SSG)

Exercise "DOM Scripting"

- Enhance the to-do list from the previous slide:
  1. when a new list element is **created**
     it will also be printed to the Web console
  2. when a list element is **clicked**
     it will be printed to the Web console
  3. … crossed out
  4. … deleted after OK on `confirm()`

- It doesn't work?
  - see Exercise "Event-Based Programming"

Forms

- defining event listeners in `init()` function
  - input validation **while** filling in the form

    ```
    inputElement.addEventListener('event', checkElement);
    ```
  - input validation **after** completing the form

    ```
    form.addEventListener('submit', checkForm);
    ```
- input elements
  - selecting

    `e.target, form.nameOfInputElement, …`
  - reading

    `value, checked, …`
- prevent form data from being sent to the server

  `e.preventDefault()`

formData.html pizzaJS.html

Exercise "JavaScript and Forms"

- Enhance your order form with input validation **while** filling in and **after** completing the form.

  Check at least one text input field, a check box and a radio button group. Put your JavaScript functions in a separate file.
- It doesn't work?
  - see Exercise "Event-Based Programming"

Input Validation for Forms

- while filling in (client – JavaScript, HTML)
  - e.g. ZIP code is a five digit number
  - `inputElement.addEventListener()`
  - events `change`, `blur`, `click`, …
- after completing, but before sending (client – JavaScript, HTML)
  - e.g. the password has been given, at least one checkbox has been checked
  - `form.addEventListener('submit',…)`
- after sending (server - PHP)
  - e.g. password is correct
  - all client-side validations again, and more checks (e.g. XSS)

pizza4u.html

Exercise "JavaScript and XSS"

- Check if your PHP implementation for the Exercise "PHP" is vulnerable to cross-site scripting.
- Fix the problem with this PHP function `https://php.net/htmlspecialchars`, e.g.:

```
$name=htmlspecialchars($_POST['theName']);
```

Intermediate Test "JavaScript"

- Implement the following behavior of a Web page in HTML, CSS, and JavaScript:

  Paragraphs become invisible when clicked.

  There is a button that makes all paragraphs visible again.

- The Web page must validate according to HTML5 without errors.

Extensible Markup Language (XML)

- **structuring**, automatic processing, communicating, storing of data/documents
- meta language to define new document types / description languages
  - application-specific tags, semantics
  - "intelligent" processing of documents
  - platform-independent, standardized
  - text-based
- Java: portable code, XML: portable data!

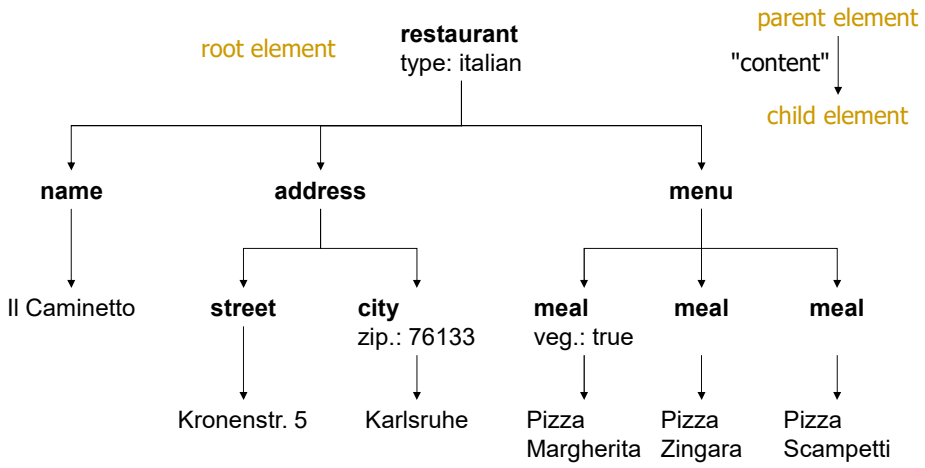
  caminetto.html

Basic Terms

		attribut name / attribut value	
markup	`<meal vegetarian="true">`		start tag
character data	`Pizza Margherita`		content
markup	`</meal>`		end tag

a (meal) element

Document Tree (Tree Structure)



created from text file (caminetto1.xml) by XML parser

XML Documents

- XML declaration

```
<?xml version="1.0" encoding="character
  code" ?>
```

- well-formed
  - always start tag with corresponding end tag
  - correct nesting of elements
  - one root element
  - attribute values in (double or single) quotation marks
  - (…)

Rendering of XML Documents

- CSS
  - in the Web

```
<?xml-stylesheet href="Stil-URL"
  type="text/css" ?>
```

restaurant.css

- XSL (Extensible Stylesheet Language)
  - XSL-FO (XSL Formatting Objects) for high-quality printing
  - XSLT (XSL Transformations) for document transformation, e.g. from any XML application to HTML

XML Applications

- application to a certain task
- is a specific markup language

- defining the language by a schema
  - schema languages: DTD, XML Schema, RELAX NG, …

- a document using a language as defined by its corresponding schema is valid

Exercise "XML Documents"

- Write down a structured data set (e.g. a concrete order, information about a movie, etc.) as an XML document.

  Check this order for well-formedness in Firefox and display its tree structure.

  Develop a CSS style and display the formatted XML document.

Dokument Type Definitions (DTDs)

- a DTD is a grammar defining the syntax of a specific markup language:
  - elements provided
  - supported structure of each element (nesting, content)
  - supported attributes of each element

- a document refers to the DTD of its language by a document type declaration

```
<!DOCTYPE root_element SYSTEM "DTD-URL">
```


Element Declaration

```
<!ELEMENT element_name (content_model)>
```

- content model
  1. text (`#PCDATA`)

  2. child elements using operators
     - `,` sequence
     - `|` choice – alternative
     - `?` choice – option
     - `*` iteration 0-n
     - `+` iteration 1-n
     - `()` grouping

Element Declaration (2)

3. mixed content
   `(#PCDATA | …)*`

4. empty elements
   `EMPTY`

Attribute Declaration

```
<!ATTLIST element_name
  attribute_name type default_value
  …
>
```

- types
  - `CDATA`
  - `NMTOKEN`
  - `(value1 | value2 | …)`
  - …

Attribute Declaration (2)

- default values
  - `#IMPLIED`
  - `#REQUIRED`
  - `"value"`

restaurant.dtd

Design Considerations

- which elements,
- structure of elements (content model)

- which attributes
- type, optional/mandatory attribute

- data as element content vs. data as attribute value

Exercise "XML Document Types"

- Develop a language for your XML document (i.e. orders, movie information, etc.). Write down the DTD (employ all operators of the content model and different kinds of attribute declarations).

  Check your XML document from the exercise "XML Documents" for validity with respect to this DTD.

Entities

- declared in the DTD

```
<!ENTITY entity_reference "replacing_text">
```

- used in the document

```
&entity_reference;
```

Parameter Entities

- declared in the DTD

```
<!ENTITY % entity_reference
"replacing_text">
```

- used in the DTD

```
%entity_reference;
```


Intermediate Test "XML"

- Write a DTD against which the following `environment` documents successfully validate. (Use all operators of the content model.) The attribute `unit` has to be specified. None of the attribute values may contain spaces.
- Draw the complete document tree of the first XML document.

```
<environment>
   <temperature>
      <sensor location="roof"
       unit="Celsius">
         17
      </sensor>
      <sensor unit="Celsius">
         15
      </sensor>
   </temperature>
   <rainfall mm="17" />
</environment>
```

```
<environment>
   <rainfall mm="0" />
</environment>
```

Objectives Revisited

- to recognize the World Wide Web as an interactive distributed system drawing on both areas, user interfaces and computer networks
- to classify the WWW as an Internet service
- to know how the Web works
- to understand the various Web technologies and to be able to employ them in an application-driven way

Applications

The end user's added value

Objectives

- to appreciate the importance of applications as the decisive factor for further development of the Internet
- to be able to think about new applications

"Information"

- publishing and retrieval
- push (user passive) vs. pull (user active)
- search engines
  - full-text search in Web pages
  - through indexing the (entire) WWW
  - two phases: visiting and indexing
- AI
  - training with the (entire) WWW
  - text, images, music, video

"Communication and Cooperation"

- Computer-Supported Cooperative Work (CSCW), groupware

	explicit	*implicit*
synchronous	IRC, A/V conferencing	shared whiteboards, shared applications (VNC, X,...), cooperation-aware applications
asynchronous	email, mailing lists, news	WWW, FTP

"Electronic Commerce"

- "making money"
- opening business processes
  - B2C, B2B
- for the customer
  - product catalogue, order processing, customer-specific pricing, promise of delivery, order status, ...
- within the system
  - session management, navigation, search facility, templates, DB connection, shopping basket, customer profiles, statistics, management, ...

Further Applications

- "Entertainment"
  - mass market
  - multi-user games, "pay-per-play"
- "Education"
  - MOOCs ("Massive Open Online Course")
  - "edutainment"
- Internet of Things, IoT
  - "Internet-Embedded Appliances", Industrial Internet, Smart Home, Healthcare, Web Cams
- …

Exercise "New Applications"

- Many applications within the Internet are adaptations of traditional processes.

  Find and describe an application that is totally new and that has become possible only through the Internet.

Objectives Revisited

- to appreciate the importance of applications as the decisive factor for further develpment of the Internet
- to be able to think about new applications

Objectives

- to recognize the Internet as a world-wide network infrastructure and to be able to explain, evaluate and exploit its services
- to appreciate the World Wide Web as an interactive distributed system and to be familiar with Web technologies
- to recognize applications as the decisive factor within the Internet and to think about new applications
- to set the context for self-guided learning on demand

Outlook

Future applications

Discussion (1)

- bandwith
- security
- intellectual properties
- information overflow
- reaching everybody
- "profitable" applications

Discussion (2)

- crowdsourcing
- any time, anywhere
- near/far vs. fast/slow net access
- further speed-up of processes
- global commerce
- cooperation vs. competition
- world-wide democracies vs. new conflicts

☞ graduates Communication and Media Engineering, Offenburg University of Applied Sciences

Q&A, Feedback

Please Fill In...

- What I liked:
- What I didn't like:
- What I would like:

Appendix A

HTML, JavaScript, CSS examples

- hello.html
- page1.html
- page2.html
- page3.html
- form.html
- date.html
- asterisks.html
- pizza.html
- style1.css
- page4.html
- style2.css
- events.html
- toDoList.html
- formData.html
- pizzaJS.html
- pizza4u.html
- pizza4u.css
- pizza4u.js

```
<!DOCTYPE html>

<!-- Web page to demonstrate HTTP requests
     T. Ruedebusch (hello.html)
-->

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Hello</title>
  </head>

  <body>
    <h2>A Link Source</h2>
    <p>Just say <a href="https://info.mi.hs-offenburg.de/~tom/world.html">
      &quot;Hello&quot;</a>...</p>

  </body>
</html>
```

```html
<!DOCTYPE html>

<!-- Web page to demonstrate basic HTML tags
     T. Ruedebusch (page1.html)
-->

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>HTML in a Nutshell</title>
  </head>

  <body>
    <h1>HTML Tags</h1>
    <p>Tags define the elements of a Web page. They are nested, resulting
      in a hierarchy of elements. This document tree structure is the content
      of the <em>Document Object Model (DOM)</em>. Tags are enclosed in pointed
      brackets, e.g. &lt;p&gt; for paragraph elements.</p>

    <h2>Headings</h2>
    <p>There are six levels of headings in HTML. The higher the number of
      the heading tag, the smaller it will be rendered. Of course, you can always
      define your own layout for headings using CSS.</p>

    <h2>Paragraphs</h2>
    <p title="I've got a title">They are frequently used in Web pages.
      This paragraph also demonstrates the use of the &quot;title&quot; attribute.
      Just move the mouse pointer over the paragraph to see what happens.</p>

    <h2>Lists</h2>
    <ol>
      <li>this is the first item of an ordered list</li>
      <li>the second item is a nested unordered list
        <!-- which is a list item itself -->
        <ul>
          <li>which has a first</li>
          <li>and a second list item</li>
        </ul>
      </li>
      <li>third item of our ordered list</li>
    </ol>
  </body>
</html>
```

```html
<!DOCTYPE html>

<!-- Web page to demonstrate basic HTML tags
     T. Ruedebusch (page2.html)
-->

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>HTML in a Nutshell</title>
  </head>

  <body>
    <h1 id="headline">HTML Tags</h1>
    <p>Tags define the elements of a Web page. They are nested, resulting
      in a hierarchy of elements. This document tree structure is the content
      of the <em>Document Object Model (DOM)</em>. Tags are enclosed in pointed
      brackets, e.g. &lt;p&gt; for paragraph elements.</p>

    <h2>Headings</h2>
    <p>There are six levels of headings in HTML. The higher the number of
      the heading tag, the smaller it will be rendered. Of course, you can always
      define your own layout for headings using CSS.</p>

    <h2>Paragraphs</h2>
    <p title="I've got a title">They are frequently used in Web pages.
      This paragraph also demonstrates the use of the &quot;title&quot; attribute.
      Just move the mouse pointer over the paragraph to see what happens.</p>

    <h2>Lists</h2>
    <ol>
      <li>this is the first item of an ordered list</li>
      <li>the second item is a nested unordered list
        <!-- which is a list item itself -->
        <ul>
          <li>which has a first</li>
          <li>and a second list item</li>
        </ul>
      </li>
      <li>third item of our ordered list</li>
    </ol>

    <h2>Preformatted Text</h2>
    <p>To avoid

      formatting          of              text

      by the browser, use the &quot;preformatted&quot; tag. White space
      (line break, space, tab) will be left intact, and a fixed-pitch
      font will be used. This is especially suitable for ASCII drawings
      (note that the browser will still recognize tags):</p>

    <hr>
    <pre>
            (  )
      Moo   (oo)
```

```
            \/-------\
             ||     | \
             ||---W||  *
             ||    ||
    </pre>
    <hr>

    <h2>Linking</h2>
    <p>Hypertext would not be hypertext without linking. So why don't we link
      to our <a href="https://www.hs-offenburg.de">University of Applied Sciences
      in Offenburg</a>, to the <a href="page1.html" target="_blank">first example
      </a> (always displayed in a new window), and to the <a href="#headline">
      beginning of this page</a>.</p>
  </body>
</html>
```

```
<!DOCTYPE html>

<!-- Web page to demonstrate basic HTML tags
     T. Ruedebusch (page3.html)
-->

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>HTML in a Nutshell</title>
  </head>

  <body>
    <h1 id="headline">HTML Tags</h1>
    <p>Tags define the elements of a Web page. They are nested, resulting
      in a hierarchy of elements. This document tree structure is the content
      of the <em>Document Object Model (DOM)</em>. Tags are enclosed in pointed
      brackets, e.g. &lt;p&gt; for paragraph elements.</p>

    <h2>Headings</h2>
    <p>There are six levels of headings in HTML. The higher the number of
      the heading tag, the smaller it will be rendered. Of course, you can always
      define your own layout for headings using CSS.</p>

    <h2>Paragraphs</h2>
    <p title="I've got a title">They are frequently used in Web pages.
      This paragraph also demonstrates the use of the &quot;title&quot; attribute.
      Just move the mouse pointer over the paragraph to see what happens.</p>

    <h2>Lists</h2>
    <ol>
      <li>this is the first item of an ordered list</li>
      <li>the second item is a nested unordered list
        <!-- which is a list item itself -->
        <ul>
          <li>which has a first</li>
          <li>and a second list item</li>
        </ul>
      </li>
      <li>third item of our ordered list</li>
    </ol>

    <h2>Preformatted Text</h2>
    <p>To avoid

      formatting          of              text

      by the browser, use the &quot;preformatted&quot; tag. White space
      (line break, space, tab) will be left intact, and a fixed-pitch
      font will be used. This is especially suitable for ASCII drawings
      (note that the browser will still recognize tags):</p>

    <hr>
    <pre>
            (  )
      Moo   (oo)
```

```
            \/-------\
             ||     | \
             ||---W||  *
             ||    ||
    </pre>
    <hr>

    <h2>Linking</h2>
    <p>Hypertext would not be hypertext without linking. So why don't we link
      to our <a href="https://www.hs-offenburg.de">University of Applied Sciences
      in Offenburg</a>, to the <a href="page1.html" target="_blank">first example
      </a> (always displayed in a new window), and to the <a href="#headline">
      beginning of this page</a>.</p>

    <h2>Images
        <a href="uasOGlg.jpg" title="click me to see a larger version of the image">
          <img src="uasOGsm.png" width="120" height="25" alt="UAS Offenburg">
        </a>
    </h2>
    <p>Since 1993, images can be embedded within an HTML page. Images
      are inline, i. e. no extra line breaks before and after. Positioning of
      images should be done using Cascading Style Sheets as this is a layout matter.
    </p>

    <p>In the heading above, you can see an image that is a link source
      (thumbnail), linking to a larger version of that image, being the link
      destination.
    </p>
  </body>
</html>
```

```
<!DOCTYPE html>

<!-- pizza order form
     form data returned by post-query
     T. Ruedebusch  (form.html)
-->

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Pizza Planet order form</title>
  </head>

  <body>
    <h2><em>Pizza Planet</em> order form
      <img src="planetSmall.png" height="100" width="100" alt="The Earth"></h2>

    <p>Welcome to the <em>Pizza Planet</em> home delivery service. Please make your
      choice. There is a 50% discount if you do not receive your pizzas within 30
      minutes.</p>

    <p>Use Order to tell us what you want.</p>

    <form method="post"
          action="http://info.mi.hs-offenburg.de/~tom/formGetPost.php">
          <!-- Warning in Firefox. Always use https: for credentials -->

      <fieldset><legend>Registered customers</legend>
        <label>Name
          <input type="text" name="theName" placeholder="Registered name here"
                 required>
        </label>
        <label>Password
          <input type="password" name="thePasswd">
        </label>
      </fieldset>

      <fieldset><legend>Pizza</legend>
        <fieldset><legend>Size</legend>
          <label>
            <input type="radio" name="theSize" value="small" required> small
          </label>
          <label>
            <input type="radio" name="theSize" value="large"> large
          </label>
          <label>
            <input type="radio" name="theSize" value="very large">
            to infinity and beyond
          </label>
        </fieldset>

        <fieldset><legend>Toppings</legend>
          <label><input type="checkbox" name="tomato" value="tomato"> tomato
          </label><br>
          <label><input type="checkbox" name="cheese" value="cheese"> cheese
          </label><br>
```

```
          <label><input type="checkbox" name="peperoni" value="peperoni"> peperoni
          </label><br>
          <label><input type="checkbox" name="sausage" value="sausage"> sausage
          </label><br>
          <label><input type="checkbox" name="tuna" value="tuna"> tuna</label>
        </fieldset>
      </fieldset>

      <p><input type="submit" value="Order">
        <input type="reset" value="Clear Form"></p>
    </form>
  </body>
</html>
```

```html
<!DOCTYPE html>

<!-- HTML page to call CGI C and PHP scripts for current date and time,
     and personalized date and time using URL argument
     T. Ruedebusch (date.html)
-->

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>CGI C and PHP scripts</title>
  </head>

  <body>
    <h3>CGI scripts in C</h3>
    <ul>
      <li><a href="https://info.mi.hs-offenburg.de/tom/cgi-bin/getdate"
        target="_blank">current date and time</a></li>
      <li><a href="https://info.mi.hs-offenburg.de/tom/cgi-bin/getmydate?Edward"
        target="_blank">personalized date and time</a></li>
    </ul>

    <h3>PHP scripts</h3>
    <ul>
      <li><a href="https://info.mi.hs-offenburg.de/~tom/getdate1.php"
        target="_blank">current date and time (HTML in PHP)</a></li>
      <li><a href="https://info.mi.hs-offenburg.de/~tom/getdate2.php"
        target="_blank">current date and time (PHP in HTML)</a></li>
    </ul>

  </body>
</html>
```

```html
<!DOCTYPE html>

<!-- HTML page to call a CGI script
     which is a modified console application program in C
     T. Ruedebusch (asterisks.html)
-->

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Asterisks triangle</title>
  </head>

  <body>
    <h2>Triangle of asterisks as a dynamically generated HTML page</h2>

    <form method="post" action="https://info.mi.hs-offenburg.de/~tom/cgi-bin/sterndr">
      <p><strong>Height: </strong> <input type="text" name="hoehe" value="5"></p>

      <p><input type="submit" value="Create page">
        <input type="reset" value="Reset"></p>

    </form>
  </body>
</html>
```

```html
<!DOCTYPE html>

<!-- pizza order form
     PHP form processing script
     T. Ruedebusch  (pizza.html)
-->

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Pizza Planet order form</title>
  </head>

  <body>
    <h2><em>Pizza Planet</em> order form
      <img src="planetSmall.png" height="100" width="100" alt="The Earth"></h2>

    <p>Welcome to the <em>Pizza Planet</em> home delivery service. Please make your
      choice. There is a 50% discount if you do not receive your pizzas within 30
      minutes.</p>

    <p>Use Order to tell us what you want.</p>

    <form method="post"
          action="https://info.mi.hs-offenburg.de/~tom/pizzaorder.php">

      <fieldset><legend>Registered customers</legend>
        <label>Name
          <input type="text" name="theName" placeholder="Registered name here">
        </label>
        <label>Password
          <input type="password" name="thePasswd">
        </label>
      </fieldset>

      <fieldset><legend>Pizza</legend>
        <fieldset><legend>Size</legend>
          <label>
            <input type="radio" name="theSize" value="small"> small
          </label>
          <label>
            <input type="radio" name="theSize" value="large"> large
          </label>
          <label>
            <input type="radio" name="theSize" value="very large">
            to infinity and beyond
          </label>
        </fieldset>

        <fieldset><legend>Toppings</legend>
          <label><input type="checkbox" name="tomato" value="tomato"> tomato</label><br>
          <label><input type="checkbox" name="cheese" value="cheese"> cheese</label><br>
          <label><input type="checkbox" name="peperoni" value="peperoni"> peperoni
          </label><br>
          <label><input type="checkbox" name="sausage" value="sausage"> sausage
          </label><br>
          <label><input type="checkbox" name="tuna" value="tuna"> tuna</label>
        </fieldset>
      </fieldset>

      <p><input type="submit" value="Order">
        <input type="reset" value="Clear Form"></p>
    </form>
  </body>
</html>
```

```
/*
     External style sheet demonstrating basic layout rules
     T. Ruedebusch (style1.css)
*/

body {
   font-family: Garamond, serif;
}

h1 {
   font-family: Arial, Helvetica, sans-serif;
   text-align: center;
}

h2 {
   font-family: Arial, Helvetica, sans-serif;
}

em {
   font-style: normal;
   color: red;
}

pre {
   background-color: rgb(220, 245, 220);
}

a {
   text-decoration: none;
   font-weight: bold;
}

img {
   float: right;
}
```

```
<!DOCTYPE html>

<!-- Web page to demonstrate basic HTML tags and styling
     T. Ruedebusch (page4.html)
-->

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>HTML in a Nutshell</title>
    <link rel="stylesheet" href="style2.css">
  </head>

  <body>
    <h1 id="headline">HTML Tags</h1>
    <p>Tags define the elements of a Web page. They are nested, resulting
      in a hierarchy of elements. This document tree structure is the content
      of the <em>Document Object Model (DOM)</em>. Tags are enclosed in pointed
      brackets, e.g. &lt;p&gt; for paragraph elements.</p>

    <h2>Headings</h2>
    <p>There are six levels of headings in HTML. The higher the number of
      the heading tag, the smaller it will be rendered. Of course, you can always
      define your own layout for headings using CSS.</p>

    <h2>Paragraphs</h2>
    <p title="I've got a title">They are frequently used in Web pages.
      This paragraph also demonstrates the use of the &quot;title&quot; attribute.
      Just move the mouse pointer over the paragraph to see what happens.</p>

    <h2>Lists</h2>
    <ol>
      <li>this is the first item of an ordered list</li>
      <li>the second item is a nested unordered list
        <!-- which is a list item itself -->
        <ul>
          <li>which has a first</li>
          <li>and a second list item</li>
        </ul>
      </li>
      <li>third item of our ordered list</li>
    </ol>

    <h2 class="compFont">Preformatted Text</h2>
    <p>To avoid

      formatting          of              text

      by the browser, use the <span class="compFont">preformatted</span> tag.
      White space (line break, space, tab) will be left intact, and a fixed-pitch
      font will be used. This is especially suitable for ASCII drawings
      (note that the browser will still recognize tags):</p>

    <hr>
    <pre>
            (   )
```

```
      Moo   (oo)
             \/-------\
              ||     | \
              ||---W||  *
              ||    ||
    </pre>
    <hr>

    <h2>Linking</h2>
    <p>Hypertext would not be hypertext without linking. So why don't we link
      to our <a href="https://www.hs-offenburg.de">University of Applied Sciences
      in Offenburg</a>, to the <a href="page1.html" target="_blank">first example
      </a> (always displayed in a new window), and to the <a href="#headline">
      beginning of this page</a>.</p>

    <h2>Images
        <a href="uasOGlg.jpg" title="click me to see a larger version of the image">
          <img src="uasOGsm.png" width="120" height="25" alt="UAS Offenburg">
        </a>
    </h2>
    <p>Since 1993, images can be embedded within an HTML page. Images
      are inline, i. e. no extra line breaks before and after. Positioning of
      images should be done using Cascading Style Sheets as this is a layout matter.
    </p>

    <p>In the heading above, you can see an image that is a link source
      (thumbnail), linking to a larger version of that image, being the link
      destination.
    </p>
  </body>
</html>
```

```
/*
     External style sheet demonstrating more layout rules
     T. Ruedebusch (style2.css)
*/

body {
   margin: 80px;                                    /* will not be inherited */
   font-family: Garamond, serif;                        /* will be inherited */
}

#headline{
   position: fixed;   /* absolutely positioned elements shrink to their size */
   top: 0;
   padding-right: 80px;
   border-bottom: purple solid 2px;
   background-color: white;
}

h1, h2 {
   font-family: Arial, Helvetica, sans-serif;
}

.compFont {
   font-family: monospace;
}

p::first-letter {
   font-size: 150%;
}

p[title] {
   background-color: gainsboro;
}

em {
   font-style: normal;
   color: red;
}

li:first-child {
   font-style: italic;
}

pre {
   background-color: rgb(220, 245, 220);
}

a {
   text-decoration: none;
   font-weight: bold;
}

h2 a {
   cursor: help;
}

a:hover {
   text-decoration: underline;
}

img {
   float: right;
}
```

```html
<!DOCTYPE html>

<!-- Adding event listeners
     T. Ruedebusch (events.html)
-->

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Events</title>
    <script>
      'use strict';

      document.addEventListener('DOMContentLoaded', init);

      function init (e) {
        document.getElementById('clickMe').addEventListener('click', switchText);
      }

      function switchText(e) {
        console.log("clicked: " + e.target);
        if (e.target.textContent == "Hello")
          e.target.textContent = "World";
        else
          e.target.textContent = "Hello";
      }

    </script>
  </head>

  <body>
    <h1 id="clickMe">Hello</h1>
  </body>
</html>
```

```html
<!DOCTYPE html>

<!-- Working with the Document Object Model
     T. Ruedebusch (toDoList.html)
-->

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Working with the DOM</title>
    <script>
      'use strict';

      document.addEventListener('DOMContentLoaded', init);

      function init () {
        let todoText = document.getElementById('todoText');
        todoText.addEventListener('change', addListItem);
      }

      function addListItem(e) {
        let todoList = document.getElementById('todoList');

        let newListItem = document.createElement('li');
        newListItem.textContent = e.target.value;
        todoList.append(newListItem);

        e.target.value = "";
      }
    </script>
  </head>

  <body>
    <h4>My To-Do List</h4>

    <input type="text" id="todoText" placeholder="to do">

    <ul id="todoList">
    </ul>

  </body>
</html>
```

```html
<!DOCTYPE html>

<!-- Accessing form data
     T. Ruedebusch (formData.html)
-->

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>My Form</title>
    <script>
    'use strict';

    document.addEventListener('DOMContentLoaded', init);

    function init () {
      let myForm = document.getElementById('myForm');
      myForm.addEventListener('submit', printFormData);
    }

    function printFormData(e) {
      let myForm = e.target;
      let output = "";
      output += "myForm.myTextField.value: " + myForm.myTextField.value + "\n\n";

      output += "myForm.myRbGroup.value: " + myForm.myRbGroup.value + "\n\n";
                                            //useful _RadioNodeList_.value

      output += "myForm.myCb.value: " + myForm.myCb.value + "\n";
      output += "myForm.myCb.checked: " + myForm.myCb.checked + "\n";

      if ( !confirm(output) )
        e.preventDefault();
    }
    </script>
  </head>

  <body>
    <form id="myForm" action="https://info.mi.hs-offenburg.de/~tom/formGetPost.php">
      <p><input type="text" id ="myTextFieldId" name="myTextField" required> </p>

      <p><input type="radio" name="myRbGroup" value="left">left
         <input type="radio" name="myRbGroup" value="middle">middle
         <input type="radio" name="myRbGroup" value="right">right </p>

      <p><input type="checkbox" name="myCb" value="yes">yes<br>

      <p><input type="submit"> <input type="reset"></p>
    </form>
  </body>
</html>
```

```
<!DOCTYPE html>

<!-- pizza order form
     JavaScript input validation
     T. Ruedebusch  (pizzaJS.html)
-->

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Pizza Planet order form</title>
  <style>
    .error {color: red;}
  </style>
    <script>
    'use strict';

    document.addEventListener('DOMContentLoaded', init);

    function init () {
      let pizzaForm = document.getElementById('pizzaForm');

    //validation while filling in the form
    pizzaForm.thePasswd.addEventListener('change', checkPasswd);

      //validation after completing the form
    pizzaForm.addEventListener('submit', checkOrder);
  }

  function checkPasswd(e) {           //validation while filling in the form
    let myPasswd = e.target;

    if (myPasswd.value.length < 8) {
      alert("password needs to have eight characters at least");
    myPasswd.value = '';
    }
  }

    function checkOrder(e) {            //validation after completing the form
      let myForm = e.target;

    if (myForm.theSize.value == '') {
      document.getElementById('sizeLegend').className = 'error';
    e.preventDefault();
    }
  }
    </script>
  </head>

  <body>
    <h2><em>Pizza Planet</em> order form
      <img src="planetSmall.png" height="100" width="100" alt="The Earth"></h2>

    <p>Welcome to the <em>Pizza Planet</em> home delivery service. Please make your
      choice. There is a 50% discount if you do not receive your pizzas within 30
      minutes.</p>

    <p>Use Order to tell us what you want.</p>

    <form method="post" id="pizzaForm"
          action="https://info.mi.hs-offenburg.de/~tom/pizzaorder.php">

      <fieldset><legend>Registered customers</legend>
        <label>Name
          <input type="text" name="theName" placeholder="Registered name here">
        </label>
        <label>Password
          <input type="password" name="thePasswd">
        </label>
      </fieldset>

      <fieldset><legend>Pizza</legend>
```

```
      <fieldset><legend id="sizeLegend">Size</legend>
        <label>
          <input type="radio" name="theSize" value="small"> small
        </label>
        <label>
          <input type="radio" name="theSize" value="large"> large
        </label>
        <label>
          <input type="radio" name="theSize" value="very large">
          to infinity and beyond
        </label>
      </fieldset>

      <fieldset><legend>Toppings</legend>
        <label><input type="checkbox" name="tomato" value="tomato"> tomato</label><br>
        <label><input type="checkbox" name="cheese" value="cheese"> cheese</label><br>
        <label><input type="checkbox" name="peperoni" value="peperoni"> peperoni
        </label><br>
        <label><input type="checkbox" name="sausage" value="sausage"> sausage
        </label><br>
        <label><input type="checkbox" name="tuna" value="tuna"> tuna</label>
      </fieldset>
    </fieldset>

    <p><input type="submit" value="Order">
      <input type="reset" value="Clear Form"></p>
  </form>
 </body>
</html>
```

```html
<!DOCTYPE html>

<!-- Pizza for you with CSS, JavaScript, PHP
     T. Ruedebusch (pizza4u.html)
-->

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="pizza4u.css">
    <title>Pizza Planet order form</title>
  </head>

  <body>
    <h2><em>Pizza Planet</em> order form
      <img src="planetSmall.png" height="100" width="100" alt="The Earth"></h2>

    <p>Welcome to the <em>Pizza Planet</em> home delivery service. Please make your
      choice. There is a 50% discount if you do not receive your pizzas within 30
      minutes.</p>

    <p>Use <span id="orderSpan">Order</span> to tell us what you want.</p>

    <form id="pizzaForm" method="post"
          action="https://info.mi.hs-offenburg.de/~tom/pizzaorder.php">

      <fieldset><legend>Registered customers</legend>
        <label class="horizontal">Name
          <input type="text" name="theName" placeholder="Registered name here"
                 required>
        </label>
        <label class="horizontal">Password
          <input type="password" name="thePasswd" id="pwField">
          <span id="peekPassword"> O</span>
        </label>
      </fieldset>

      <fieldset><legend>Pizza</legend>
        <fieldset><legend>Size</legend>
          <label class="horizontal">
            <input type="radio" name="theSize" value="small" required> small
          </label>
          <label class="horizontal">
            <input type="radio" name="theSize" value="large"> large
          </label>
          <label class="horizontal">
            <input type="radio" name="theSize" value="very large">
            to infinity and beyond
          </label>
        </fieldset>

        <fieldset><legend>Toppings</legend>
          <label><input type="checkbox" name="tomato" value="tomato"> tomato
          </label><br>
          <label><input type="checkbox" name="cheese" value="cheese"> cheese
          </label><br>
```

```
          <label><input type="checkbox" name="peperoni" value="peperoni"> peperoni
          </label><br>
          <label><input type="checkbox" name="sausage" value="sausage"> sausage
          </label><br>
          <label><input type="checkbox" name="tuna" value="tuna"> tuna</label>
        </fieldset>
      </fieldset>
      <fieldset id="yourOrderFs"><legend>Your Order</legend>
        <div id="currentOrder"></div>
      </fieldset>

      <p><input type="submit" value="Order">
        <input type="reset" value="Clear Form"></p>
    </form>

    <script src="pizza4u.js"></script>
  </body>
</html>
```

```
/*
     Styling our pizza order form
     T. Ruedebusch (pizza4u.css)
*/


.noOrder {display: none;}
.validOrder {display: block;}

body {
   margin: 5em;
   font-family: Garamond, serif;
   background-color: whiteSmoke;
}

em {
   font-style: normal;
   color: gray;
}

img {
   float: right;
   padding: .2em;
}

#orderSpan {
   border: 1px solid black;
   background-color: white;
   padding-left: .1em;
   padding-right: .2em;
}

fieldset{
   background-color: gainsboro;
   border-style: none;
   padding-bottom: 1em;
}

fieldset fieldset, input:hover {
   background-color: whiteSmoke;
}

legend {
   color: gray;
   font-weight: bold;
   font-size: 150%;
}

fieldset fieldset legend {
   font-size: 120%;
}

input {
   border: 1px solid black;
   padding: .2em;
   background-color: white;
```

```
}

input[type="text"], input[type="password"] {
   width: 11em;
}

input[type="submit"], input[type="reset"] {
   width: 8em;
}

label.horizontal {
   display: inline-block;
   width: 11em;
}

label:hover {
   font-weight: bold;
}
```

```
/*   Checking and refreshing our pizza order form
     T. Ruedebusch (pizza4u.js) */

'use strict';

document.addEventListener('DOMContentLoaded', init);
document.addEventListener('DOMContentLoaded', refreshOrder);//make reload consistent

function init () {
  let pizzaForm = document.getElementById('pizzaForm');
  pizzaForm.addEventListener('submit', checkOrder);
  pizzaForm.addEventListener('reset', function ()            //make reset consistent
    {document.getElementById('yourOrderFs').className = "noOrder";});

  let pizzaSpecs = document.querySelectorAll('input[type="radio"], \
                                              input[type="checkbox"]');
  for (let i = 0; i < pizzaSpecs.length; i++)
    pizzaSpecs[i].addEventListener('change', refreshOrder);

  document.getElementById('peekPassword').addEventListener('click', function(e) {
    let pwField = document.getElementById('pwField');
    if (pwField.type == "password")
      pwField.type = "text";
    else
      pwField.type = "password";
    e.preventDefault();                            //don't select/highlight the text
  });
}

function getToppings() {
  let toppingsCheckboxes = document.querySelectorAll('input[type="checkbox"]');
  let toppings = "";

//  for (let i = 0; i < toppingsCheckboxes.length; i++)
//    if (toppingsCheckboxes[i].checked)
//      toppings += toppingsCheckboxes[i].value + " ";

  toppingsCheckboxes.forEach( function(item)                //replaces for loop above
    {if (item.checked) toppings += item.value + " ";} );

  return toppings;
}

function checkOrder(e) {
  let toppings = getToppings();

  if (toppings == "")
    if (!confirm("No toppings at all?!"))
      e.preventDefault();
}

function refreshOrder() {        //actual form data with form.nameOfRadiobuttonGroup
  let theSize = document.getElementById('pizzaForm').theSize.value;
  let toppings = getToppings();

  let orderText = "";
```

```
  if (theSize != "" || toppings != "")
    orderText = "One " + theSize + " pizza ";
  if (toppings != "")
    orderText += "with " + toppings;

  if (orderText != "") {
    document.getElementById('currentOrder').textContent = orderText;
    document.getElementById('yourOrderFs').className = "validOrder";
  }
  else
    document.getElementById('yourOrderFs').className = "noOrder";
}
```

Appendix B

CGI/C and PHP examples

- `helloWorld.c`
- `getdate.c`
- `getmydate.c`
- `getdate1.php`
- `getdate2.php`
- `pizzaorder.php`
- `stateURL.php`
- `stateForm.php`
- `stateFormHidden.php`
- `stateCookie.php`
- `toDoList.php`

```c
/* CGI script as a first example (helloWorld.c)                          *
 * T. Ruedebusch                                                         */

#include <stdio.h>                        /* printf() */

int main()
{
   printf("Content-Type: text/html\n\n"); /* note double \n */

   printf("<!DOCTYPE html>\n");

   printf("<html lang='en'><head>\n");    /* note quotation marks */
   printf("<meta charset='UTF-8'>\n");
   printf("<title>Hello World</title></head>\n");

   printf("<body>\n");
   printf("<h2>Hello World!</h2>\n");
   printf("</body></html>\n");
   return 0;
}
```

```c
/* CGI script to show current date and time (getdate.c)                    *
 * makes use of 'date' shell command                                       *
 * T. Ruedebusch                                                           */

#include <stdio.h>                          /* printf() */
#include <stdlib.h>                         /* system() */

int main()
{
   printf("Content-Type: text/html\n\n"); /* note double \n */

   printf("<!DOCTYPE html>\n");

   printf("<html lang='en'><head>\n");    /* note quotation marks */
   printf("<meta charset='UTF-8'>\n");
   printf("<title>Current date</title></head>\n");

   printf("<body>\n");
   printf("<h3>Current date</h3>\n");
   printf("<p>The current date is ");

   fflush(stdout);                         /* write all text before... */
   system("date");                         /* call date from shell*/

   printf("</p></body></html>\n");
   return 0;
}
```

```c
/* CGI script to show current date and time (getmydate.c)                  *
 * with individual name via one single URL argument                        *
 * makes use of 'date' shell command                                       *
 * T. Ruedebusch                                                           */

#include <stdio.h>                        /* printf() */
#include <stdlib.h>                       /* system(), getenv() */

int main()
{
   printf("Content-Type: text/html\n\n"); /* note double \n */

   printf("<!DOCTYPE html>\n");

   printf("<html lang='en'><head>\n");    /* note quotation marks */
   printf("<meta charset='UTF-8'>\n");
   printf("<title>Current date</title></head>\n");

   printf("<body>\n");
   printf("<h3>Current date</h3>\n");
   printf("<p>The current date is ");

   fflush(stdout);                        /* write all text before... */
   system("date");                        /* call date from shell*/

   printf(", %s.", getenv("QUERY_STRING")); /* get and print single URL arg */

   printf("</p></body></html>\n");
   return 0;
}
```

```php
<?php
/* PHP script to show current date and time (getdate1.php)                    *
 * makes use of date() function                                               *
 * "HTML in PHP"                                                              *
 * T. Ruedebusch                                                              */

   echo "<!DOCTYPE html>";

   echo "<html lang='en'><head>\n";                      /*note quotation marks*/
   echo "<meta charset='UTF-8'>";
   echo "<title>Current date</title></head>\n";

   echo "<body>\n";
   echo "<h3>Current date</h3>\n";
   echo "<p>The current date is ";

   echo date("D M d H:i:s T Y")."\n";                    /*formatted date*/

   echo "</p></body></html>\n";
?>
```

```
<!DOCTYPE html>

<!-- PHP script to show current date and time (getdate2.php)
     makes use of date() function
     "PHP in HTML"
     T. Ruedebusch
-->
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Current date</title>
  </head>

  <body>
    <h3>Current date</h3>
    <p>The current date is
      <?php
        echo date("D M d H:i:s T Y")."\n";                        /*formatted date*/
      ?>
    </p>

  </body>
</html>
```

```
<!DOCTYPE html>

<!-- PHP script to process the pizza order form (pizzaorder.php)
     makes use of mail() function
     T. Ruedebusch
-->

<?php /* all client-side checks have to be done again! */
   if(!empty($_POST['theName'])) //includes isset()
      $name =  $_POST['theName'];
   else
      $name = "Foobar";

   if(isset($_POST['theSize']))
      $size =  $_POST['theSize'];
   else
      $size = "";

   $toppings = "";
   if(isset($_POST['tomato'])) $toppings .= "tomato, ";
   if(isset($_POST['cheese'])) $toppings .=  "cheese, ";
   if(isset($_POST['peperoni'])) $toppings .=  "peperoni, ";
   if(isset($_POST['sausage'])) $toppings .=  "sausage, ";
   if(isset($_POST['tuna'])) $toppings .=  "tuna, ";
?>

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="pizza4u.css">
    <title>Pizza Planet order confirmation</title>
  </head>

  <body>
    <h3>Hello, <?php echo $name ?>!</h3>

    <p>Within the next 30 minutes, you will receive your
       <?php echo "$size pizza with $toppings"; ?>
       and our famous cheesy crust.</p>

    <?php
      if(substr($_SERVER['REMOTE_ADDR'], 0, 7) == "141.79.") //UAS OG domain
        echo "<p>Free Coke for on-campus deliveries!</p>";
    ?>

  </body>
</html>

<?php /* send order to pizza baker */
   mail("ruedebusch@hs-offenburg.de", "new pizza order",
        "Make one $size pizza with $toppings for $name.");
?>
```

```
<!DOCTYPE html>

<!-- PHP script passing state via URL (stateURL.php)
     T. Ruedebusch
-->

<?php
   if(!empty($_SERVER['QUERY_STRING']))
      $number = $_SERVER['QUERY_STRING'] + 1;
   else
      $number = 1;
?>

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>PHP script passing state via URL</title>
  </head>

  <body>

    <h2>This is the <?php echo $number; ?>. Script Call</h2>

    <h3>Call it
      <a href="stateURL.php?<?php echo $number; ?>"> again</a>
    </h3>

  </body>
</html>
```

```
<!DOCTYPE html>

<!-- PHP script passing state via text input field (stateForm.php)
     T. Ruedebusch
-->

<?php
   if(isset($_POST['number']))
      $number = $_POST['number'] + 1;
   else
      $number = 1;
?>

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>PHP script passing state via text input field</title>
  </head>

  <body>
    <form method="post" action="stateForm.php">

      <h2>This is the
        <input type="text" size="1" name="number"
               value="<?php echo $number; ?>">. Script Call
      </h2>

      <h3>Call it <input type="submit" value="again"></h3>

    </form>
  </body>
</html>
```

```
<!DOCTYPE html>

<!-- PHP script passing state via hidden field (stateFormHidden.php)
     T. Ruedebusch
-->

<?php
   if(isset($_POST['number']))
      $number = $_POST['number'] + 1;
   else
      $number = 1;
?>

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>PHP script passing state via hidden field</title>
  </head>

  <body>
    <form method="post" action="stateFormHidden.php">

      <h2>This is the <?php echo $number; ?>. Script Call</h2>

      <h3>Call it <input type="submit" value="again"></h3>

      <input type="hidden" name="number" value="<?php echo $number; ?>">

    </form>
  </body>
</html>
```

```
<!DOCTYPE html>

<!-- PHP script passing state via cookie (stateCookie.php)
     T. Ruedebusch
-->

<?php
   if(isset($_COOKIE['number']))
      $number = $_COOKIE['number'] + 1;
   else
      $number = 1;

   setcookie('number', $number, time()+(86400*365),'/','mi.hs-offenburg.de',true);
?>

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>PHP script passing state via cookie</title>
  </head>

  <body>

    <h2>This is the <?php echo $number; ?>. Script Call</h2>

    <h3>Call it <a href="stateCookie.php">again</a></h3>

  </body>
</html>
```

```
<!DOCTYPE html>

<!-- To-Do list in PHP, using sessions
     T. Ruedebusch
-->

<?php
   session_start();

   if(empty($_GET['newItem']))
      $_SESSION['items'] = array();
   else
      array_push($_SESSION['items'], $_GET['newItem']);

?>

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>To-Do list in PHP</title>
  </head>

  <body>
    <h4>My To-Do List</h4>

    <form action="toDoList.php">
      <input type="text" id="todoText" placeholder="to do" name="newItem">
    </form>

    <ul id="todoList">

<?php
   for ($i = 0; $i < count($_SESSION['items']); $i++)  // or foreach()
      echo "<li>" . $_SESSION['items'][$i] . "</li>\n";
?>

    </ul>

  </body>
</html>
```

Appendix C

XML examples

- `caminetto.html`
- `caminetto1.xml`
- `caminetto2.xml`
- `caminetto3.xml`
- `restaurant.css`
- `restaurant.dtd`

```html
<!DOCTYPE html>

<!-- Very simple Caminetto homepage
     T. Ruedebusch (caminetto.html)
-->
<html lang="de">
  <head>
    <meta charset="UTF-8">
    <title>Il Caminetto</title>
  </head>

  <body>

    <h1>Il Caminetto</h1>

    <p>Kronenstra&szlig;e 5 in
      Karlsruhe</p>

    <ul>
      <li>Pizza Margherita</li>
      <li>Pizza Zingara</li>
      <li>Pizza Scampetti</li>
    </ul>

  </body>
</html>
```

```xml
<?xml version="1.0" encoding="ISO-8859-15" ?>

<!-- Favorite Italian restaurants
   T. Ruedebusch (caminetto1.xml)
-->

<restaurant type="italian">

  <name>Il Caminetto</name>

  <address>
    <street>Kronenstraße 5</street>
    <city zipcode="76133">Karlsruhe</city>
  </address>

  <menu>
    <meal vegetarian="true">Pizza Margherita</meal>
    <meal>Pizza Zingara</meal>
    <meal>Pizza Scampetti</meal>
  </menu>

</restaurant>
```

```xml
<?xml version="1.0" encoding="ISO-8859-15" ?>
<?xml-stylesheet href="restaurant.css" type="text/css" ?>

<!-- Favorite Italian restaurants with CSS style
   T. Ruedebusch (caminetto2.xml)
-->

<restaurant type="italian">

  <name>Il Caminetto</name>

  <address>
    <street>Kronenstraße 5</street>
    <city zipcode="76133">Karlsruhe</city>
  </address>

  <menu>
    <meal vegetarian="true">Pizza Margherita</meal>
    <meal>Pizza Zingara</meal>
    <meal>Pizza Scampetti</meal>
  </menu>

</restaurant>
```

```xml
<?xml version="1.0" encoding="ISO-8859-15" ?>
<?xml-stylesheet href="restaurant.css" type="text/css" ?>

<!DOCTYPE restaurant SYSTEM "restaurant.dtd">

<!-- Favorite Italian restaurants with CSS style and validation
   T. Ruedebusch (caminetto3.xml)
-->

<restaurant type="italian">

  <name>Il Caminetto</name>

  <address>
    <street>Kronenstraße 5</street>
    <city zipcode="76133">Karlsruhe</city>
  </address>

  <menu>
    <meal vegetarian="true">Pizza Margherita</meal>
    <meal>Pizza Zingara</meal>
    <meal>Pizza Scampetti</meal>
  </menu>

</restaurant>
```

```
/*
     CSS Style sheet for restaurant documents
     T. Ruedebusch (restaurant.css)
*/

restaurant {
   font-family: Garamond, serif;
}

name, address, menu {
   display: block;
}

name {
   font-family: Arial, Helvetica, sans-serif;
   font-size: 20pt;
}

name:hover {
   color: red;
}

meal {
   margin-top: 4pt;
   display: list-item;
   list-style: disc inside;
}

meal[vegetarian="true"] {
   color: green;
}
```

```
<!-- DTD for restaurant documents
     T. Ruedebusch (restaurant.dtd)
-->

<!ELEMENT restaurant (name, address+, menu?)>

<!ATTLIST restaurant
  type          CDATA           #REQUIRED
  rating        CDATA           #IMPLIED
>

<!ELEMENT name (#PCDATA)>

<!ELEMENT address (gpscoordinates | (street, city))>

<!ELEMENT gpscoordinates EMPTY>

<!ATTLIST gpscoordinates
  longitude     NMTOKEN         #REQUIRED
  latitude      NMTOKEN         #REQUIRED
>

<!ELEMENT street (#PCDATA)>

<!ELEMENT city (#PCDATA)>

<!ATTLIST city
  zipcode       NMTOKEN         #IMPLIED
>

<!ELEMENT menu (meal*)>

<!ELEMENT meal (#PCDATA)>

<!ATTLIST meal
  vegetarian    (true|false)    "false"
>
```