Tutorium Skriptsprachen

2009 - Max Maurer

Ein Witz

> “
>
> echo "Your stdio isn't very std."
>
> -- Larry Wall in Configure from the perl distribution
>
> ”

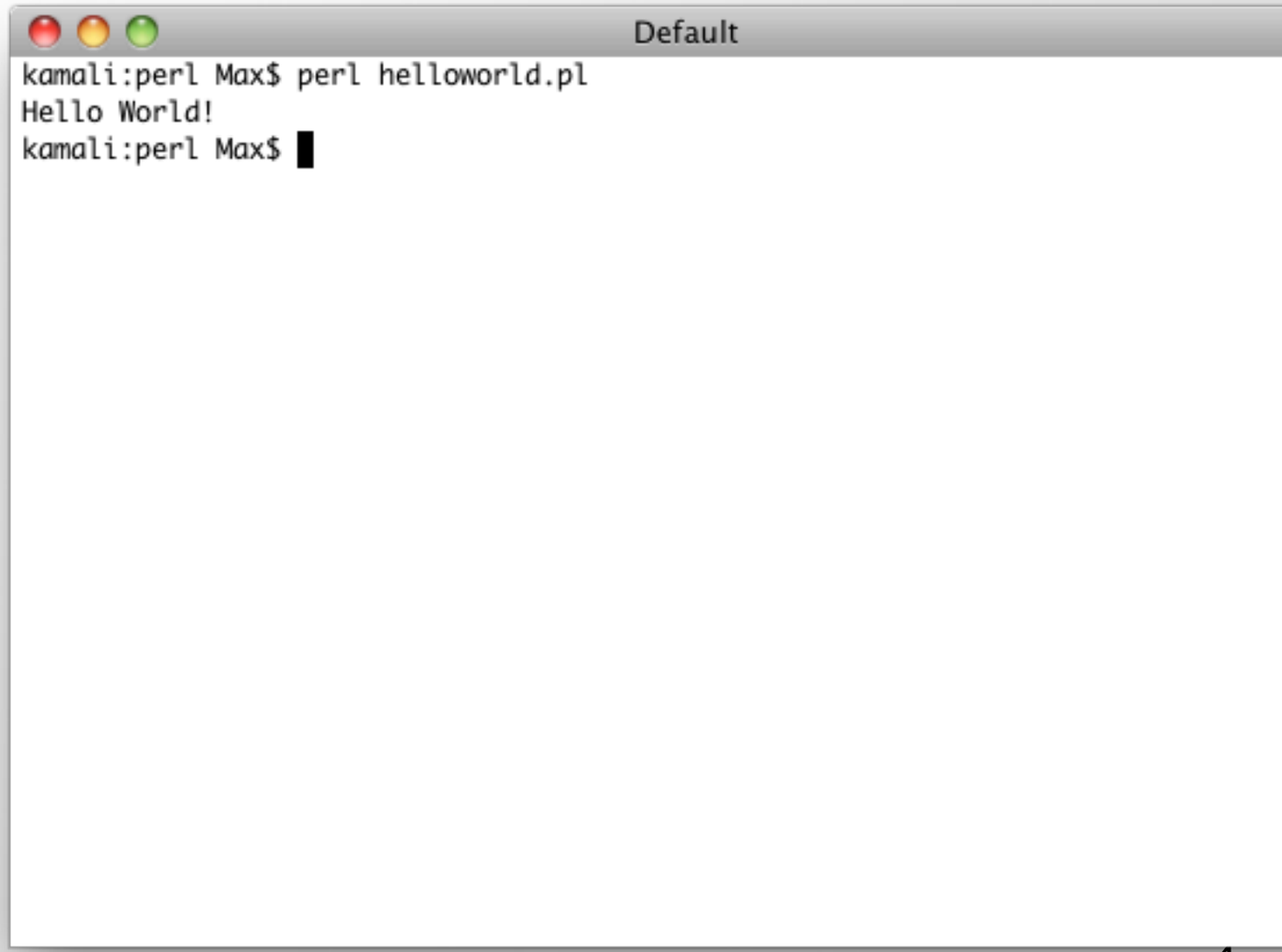
http://www.cpan.org/misc/lwall-quotes.txt.gz

.pl Perl

Hello World

Hello World!

```
#!/path/to/perl
print "Hello, World!"
```



.pl Perl

Allgemeines

Fakten

Entstanden:	1987
Erfinder:	Larry Wall
Firma:	-
Lizenz:	GPL, artistic license
Stärken:	Regexes
Anwendungsgebiete:	Serverskripte

Larry Wall

- Der Freak!
- geboren 1954 in Los Angeles
- Lingustik Studium
- Erfinder von Perl, patch, rn und metaconfig
- zweifacher Sieger des IOCCC-Wettbewerbs für den ‚undurchsichtigsten Code‘
- spielt leidenschaftlich Geige
- aber... verheiratet + 4 Kinder

Einsatzgebiete

- Perl fast überall unbemerkt im Einsatz
- als .cgi oder durch url-rewriting
- Große Firmen benutzen Perl:
  - Amazon.com
  - BBC.co.uk
  - imdb.com

Characteristika

- Interpreterbasierte Sprache
- mehrere Möglichkeiten Dinge zu tun (TIMTOWTDI)
  Beispiel: && vs and
- prozedural, modular, nur teilweise objektorientiert
- dynamische und **schwache** Typen
- Perl hat noch keinen Garbage Collector sondern benutzt reference counting
- Gui mit Perl/Tk

.pl Perl

Code-Beispiele

Variablendeklaration x=0

- Drei Kennzeichnungen von Variablen

$	Zahlen, Zeichektten, Objekte, Strings
@	Array (als Ganzes)
%	Hashes (als Ganzes)

```
#!/usr/bin/perl
my $a = "Hallo";
print $a."\n";
my @array = (3,4,5);
print $array[0]."\n";
my %hash = ("schlüssel", "wert");
print $hash{"schlüssel"}."\n";
```

```
Default
schiller:perl Max$ perl variables.pl
Hallo
3
wert
schiller:perl Max$
```

Funktionen und Parameter

- Funktionsdefinitionen mit Schlüsselwort sub
- Parameter werden mit Hilfe der Spezialvariablen @_ übergeben
- Andere Spezialvariablen $_ müssen teilweise gar nicht erst genannt werden

```
#!/usr/bin/perl
sub sum {
    my ($a1, $a2) = @_;
    return $a1+$a2;
}

print sum(1,2)."\n";
```

```
Default
schiller:perl Max$ perl sum.pl
3
schiller:perl Max$
```

Spezialvariablen

√

- Neben @_ auch andere wie $_, $1, $2, $3 usw.
- $_ kann teilweise sogar weg gelassen werden

```
#!/usr/bin/perl
open (FH, "<test.txt");
while ($line = <FH>) {
    print "\t";
  if ($line =~ /Perl/) {
    print "Magic!\n";
    } else {
      print uc $line;
    }
}
print "The same is:\n";
open (FH, "<test.txt");
while (<FH>) {
  print "\t".(/Perl/?"Magic!\n":uc);
}
```

```
some
lines
in
Perl
a
text
file
```

```
schiller:perl Max$
        SOME
        LINES
        IN
        Magic!
        A
        TEXT
        FILE
The same is:
        SOME
        LINES
        IN
        Magic!
        A
        TEXT
        FILE
schiller:perl Max$
```

Variables and sco(o)p(e)s

- Fürher kein Schlüsselwort zur Variablendeklaration notwendig
- Heute gibt es my, our und lokal

```
#!/usr/bin/perl -wT
sub declare {
    my $a = "a";
    our $b = "b"; # globally visible
    local $c = "c";
    # a ends here
    func();
    # c ends here
}

sub func() {
    print $c."\n";
}

declare();
print $b."\n";
```

Default

```
schiller:perl Max$ perl -wT our.pl
c
b
schiller:perl Max$
```

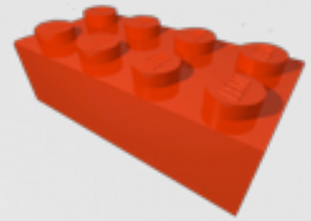
Error Handling

- Keine wirklich Fehlerbehandlung (könnte aber simuliert werden)
- Normalerweise arbeitet man mit Rückgabewerten

```
#!/usr/bin/perl
open (FH,">test.txt") || die "Did not work: $!\n";
```

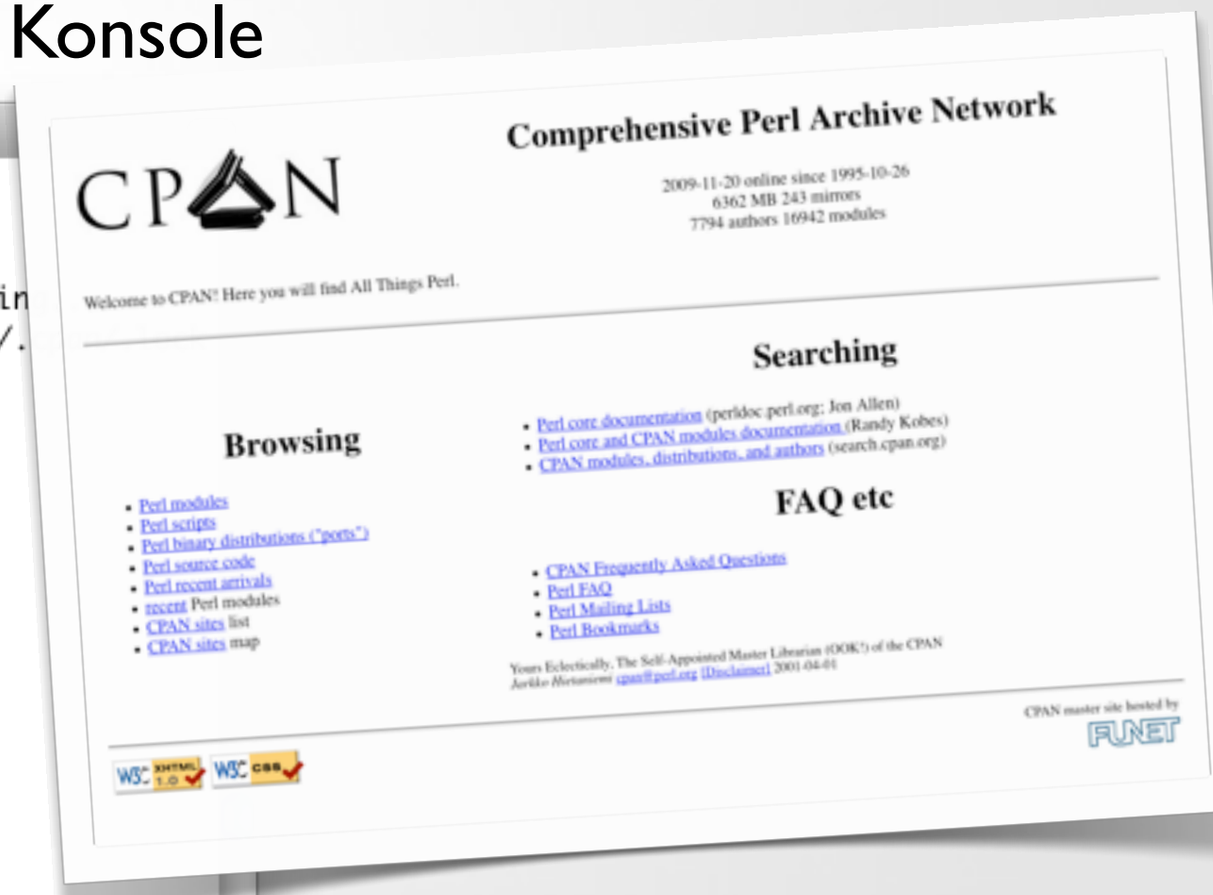
```
Default
schiller:perl Max$ perl exceptions.pl
Did not work: Permission denied
schiller:perl Max$
```

Module



- Sehr viele Module für Perl (Datenbanken, Dateitypen: PDF, MP3, DOC, XLS und vieles mehr)
- Alle an zentraler Stelle verfügbar (CPAN)
- Installation einfach über die Konsole

```
Default
kamali:perl Max$ sudo perl -MCPAN -e shell
Password:

There seems to be running another CPAN process (pid 54945).  Contactin
Other job not responding. Shall I overwrite the lockfile '/Users/Max/.
'? (Y/n) [y] y

cpan shell -- CPAN exploration and modules installation (v1.9301)
ReadLine support enabled

cpan[1]> install Mail::Webmail::Gmail
CPAN: Storable loaded ok (v2.21)
Going to read /Users/Max/.cpan/Metadata
  Database was generated on Fri, 20 Nov 2009 18:27:04 GMT
Running install for module 'Mail::Webmail::Gmail'
CPAN: YAML loaded ok (v0.70)
Running make for M/MI/MINCUS/Mail-Webmail-Gmail-1.09.tar.gz
CPAN: Time::HiRes loaded ok (v1.9719)
  LWP not available

Trying with "/opt/local/bin/curl -L -f -s -S --netrc-optional" to get
    http://www.perl.org/CPAN/authors/id/M/MI/MINCUS/Mail-Webmail-Gmail-1.09.tar.
gz
CPAN: Compress::Zlib loaded ok (v2.023)
```



Eigene Module

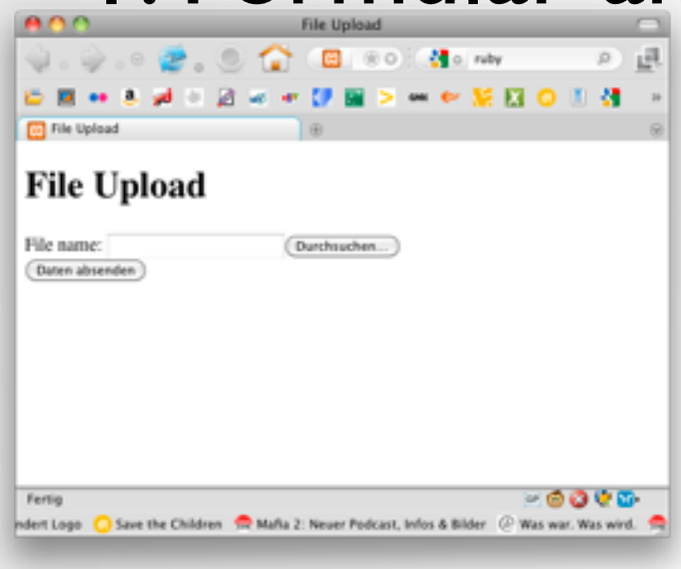
- Basically a normal Perl file...
- ... but with some special things

```
=head1 NAME

NewModule - Perl module for hooting
=cut


package NewModule;

use strict;
use vars qw($VERSION @ISA @EXPORT @EXPORT_OK);

require Exporter;

@ISA = qw(Exporter AutoLoader);
# Items to export into callers namespace by default. Note: do not export
# names by default without a very good reason. Use EXPORT_OK instead.
# Do not simply export all your public functions/methods/constants.
@EXPORT = qw(

);
$VERSION = '0.01';
```

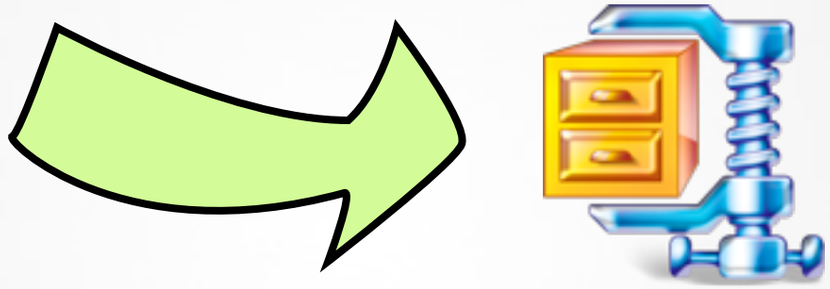
Die Aufgabe

Galerie Baukasten

1. Formular anzeigen

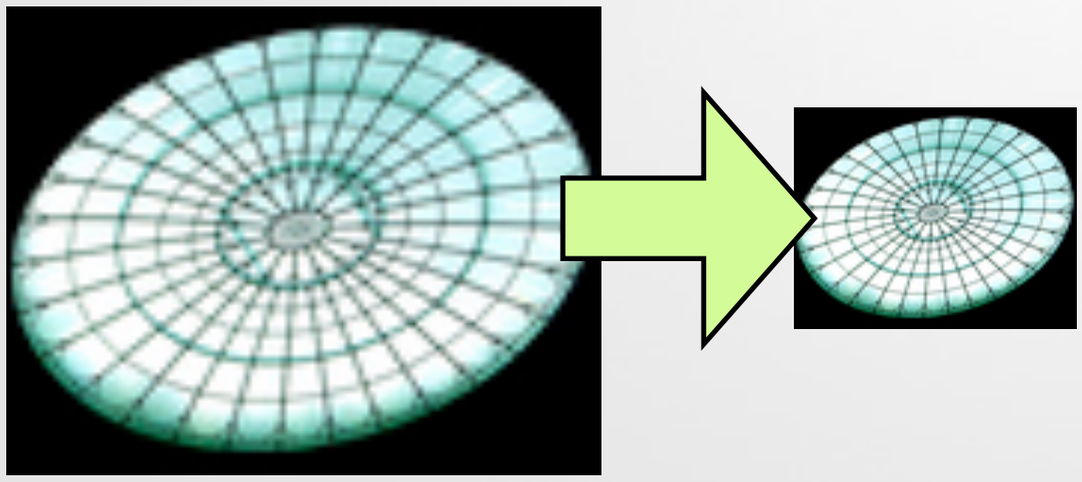


2. Zip-Datei hochladen



3. Entpacken

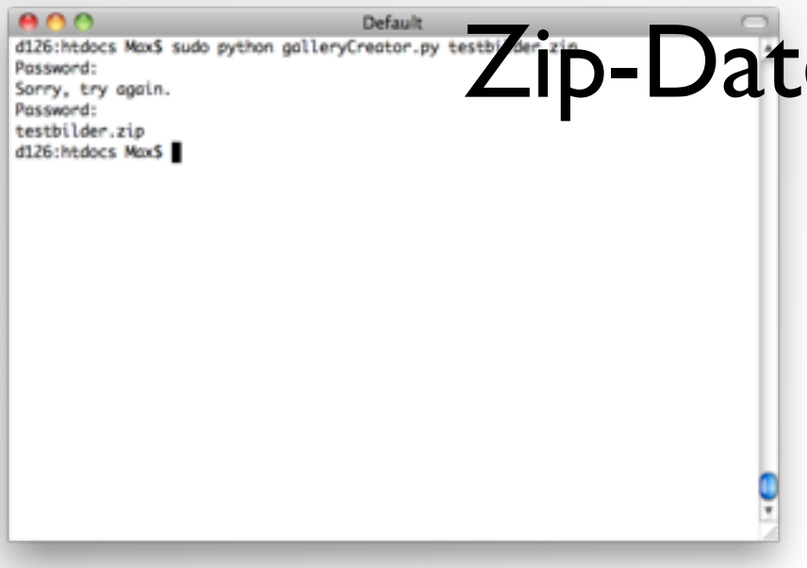
4. Thumbnails rendern



5. Gallerie Seite anzeigen

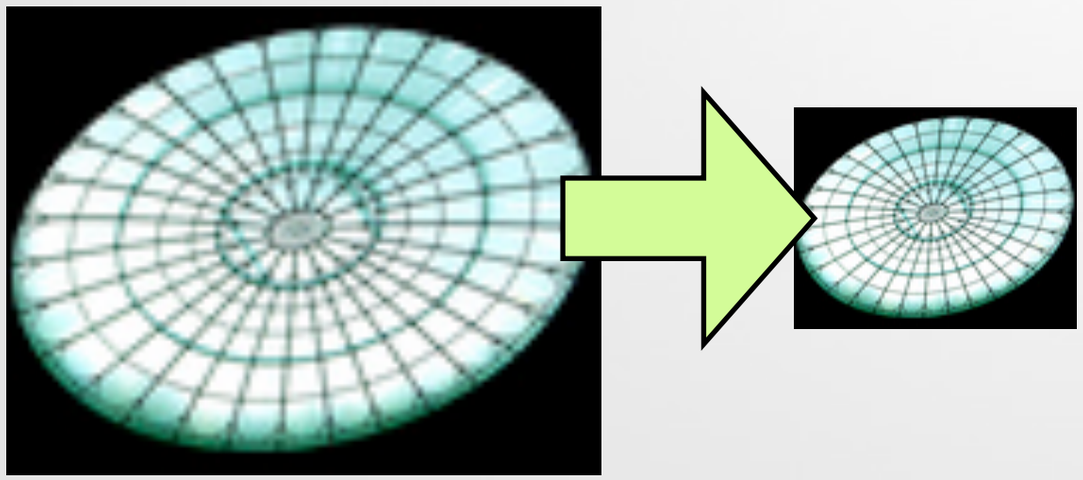
Kommandozeilenversion

1. Kommandozeilenaufruf mit Zip-Datei



2. Entpacken

3. Thumbnails rendern



4. Gallerie Seite erzeugen

Grundskript und Kommandozeile

Grundskript

```
#!/usr/bin/perl
use CGI qw(:cgi);
our $cgi = new CGI;

sub errorPage {
    my ($text) = @_;
    print $cgi->header();
    print qq{<html><body><h1>$text</h1></body></html>};
    exit()
}

if ($#ARGV >= 1) {
    # wir haben ein command line argument!
    print "Command line\n";
} else {
    print $cgi->header();
    print $HTML_TEMPLATE;
}
```

Das ‚cgi'-Modul

- „CGI.pm is a stable, complete and mature solution for processing and preparing HTTP requests and responses."
- Wichtige Parameter und Funktionen
  - $cgi->header: Gibt den Header-Teil des HTTP-requests aus inklusive eventueller cookies, redirects oder anderer Informationen
  - Einfach HTML-Erzeugung: $cgi->h1 oder $cgi->start_html
  - Zugriff auf Parameter aus Formularen $cgi->param(‚feld')

Grundskript

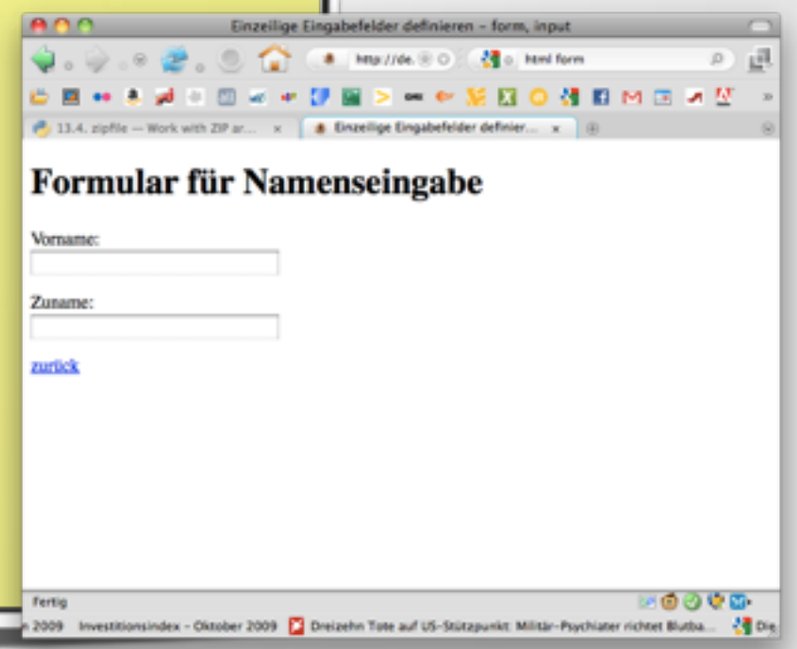
```
#!/usr/bin/perl
use CGI qw(:cgi);
our $cgi = new CGI;

sub errorPage {
    my ($text) = @_;
    print $cgi->header();
    print qq{<html><body><h1>$text</h1></body></html>};
    exit()
}

if ($#ARGV >= 1) {
    # wir haben ein command line argument!
    print "Command line\n";
} else {
    print $cgi->header();
    print $HTML_TEMPLATE;
}
```

HTML-Formulare

HTML-Formulare

- Darstellung verschiedener Eingabemöglichkeiten
  - Eingabefelder, Dropdown, Checkbox, Radio Button, Datei Upload, TextArea, Button

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Einzeilige Eingabefelder definieren</title>
</head>
<body>

<h1>Formular f&uuml;r Namenseingabe</h1>

<form action="input_text.htm" method="post">
  <p>Vorname:<br><input name="vorname" type="text" size="30"
maxlength="30"></p>
  <p>Zuname:<br><input name="zuname" type="text" size="30"
maxlength="40"></p>
</form>
```



Formulardaten allgemein

- Zwei Methoden: GET oder POST
- GET
  - Übergabe über die URL: „test.py?action=hallo&var1=wert1“
  - Direkt sichtbar und manipulierbar. Variablen bleiben bei Copy&Paste in E-Mails z.B. erhalten (z.B. Google Maps)
- POST
  - Nicht im Browser sichtbar auch nicht im Browser Cache gespeichert, werden im Anfrage-Header von HTTP übergeben

Formulardaten in Perl

- POST Formulare maskieren eventuelle GET-Werte
- Lösung: Entweder mit Hidden-Feldern arbeiten oder die GET-Werte selbst parsen

```
use CGI qw(:cgi);
our $cgi = new CGI;
print $cgi->param(‚test1‘)
```

Modul „CGI::Carp“

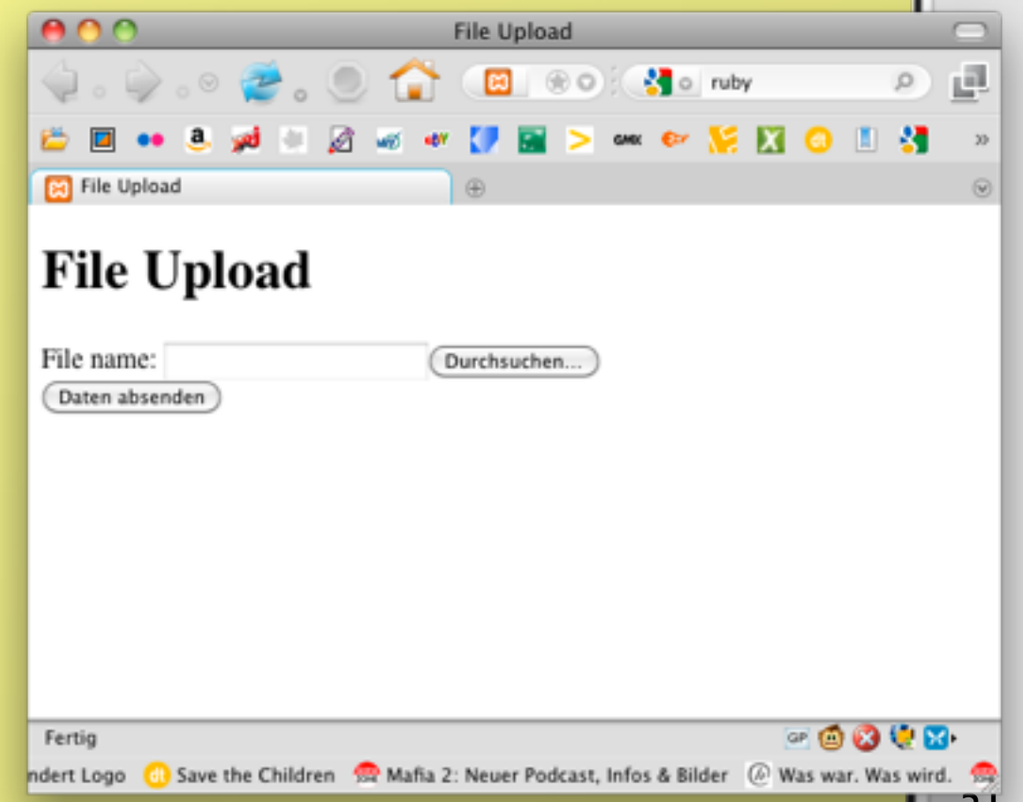
- Ausgabe von Fehlermeldungen und Stacktrace im Browser
- Nicht ganz so detailliert wie bei Python aber ähnlich

```
use CGI::Carp qw(fatalsToBrowser);
BEGIN {
  # Stacktrace anzeigen. Auch für warnings
  $SIG{__DIE__} = sub { CGI::Carp::confess @_ };
  $SIG{__WARN__} = sub { CGI::Carp::confess @_;
exit; };
}
```

Datei-Upload

Formular ausgeben

```
our $HTML_TEMPLATE = qq{
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<html><head><title>File Upload (Perl)</title>
</head><body><h1>File Upload (Perl)</h1>
<form action="" method="post" enctype="multipart/form-data">
<input type="hidden" name="action" value="upload"/>
File name: <input name="file" type="file"/><br/>
<input name="submit" type="submit"/>
</form>
</body>
</html>
};

   print $cgi->header();
   print $HTML_TEMPLATE;
```



Upload auslesen

- CGI-Modul kennt handle auf die Datei
- Muss nun gespeichert werden

```
use CGI qw(:cgi);
our $cgi = new CGI;

sub processFile {
   my ($form_field) = @_;
   $fileitem = $cgi->param($form_field);
   $filepath = $UPLOAD_DIR."/".$fileitem;
   open DAT,">$filepath" or die 'Error processing file: ',$!;
   # Dateien in den Binaer-Modus schalten
   binmode $filepath;
   binmode DAT;
   my $data;
   while(read $fileitem,$data,1024) {
     print DAT $data;
   }
   close DAT;
   return $filepath;
}
```

ZIP-Datei entpacken

Zugriff auf ZIP-Dateien

- Entpacken bietet bereits Möglichkeit den Pfad weg zu lassen

```
use Archive::Zip qw( :ERROR_CODES :CONSTANTS );

sub unzip {
    my ($file, $path) = @_;
  $zip = Archive::Zip->new();
  $zip->read($file);
  my @members = $zip->members();
  foreach (@members) {
    my $file=$_->fileName();
    if ($file=~m/.*\/(.*)$/) {
      $file = $1;
    }
    if ($file) {
      $zip->extractMemberWithoutPaths($_,$path."/".$file);
    }
  }
}
```

Das ‚Archive::Zip'-Modul

- Umgang mit ZIP-Dateien
- Wichtige Parameter und Funktionen
  - Archive::Zip->new(): Konstruktor
  - $zip->read(datei) liest vorhandene Zip-Datei ein
  - $zip->addDirectory(dir) legt ein Verzeichnis an
  - $zip->addFile(file) packt eine Datei von der Festplatte ins Archiv
  - $zip->writeToFileNamed($fileName) speichert die Datei schließlich ab

Zugriff auf ZIP-Dateien

- Hier **keine** regulären Ausdrücke notwendig

```
use Archive::Zip qw( :ERROR_CODES :CONSTANTS );

sub unzip {
    my ($file, $path) = @_;
  $zip = Archive::Zip->new();
  $zip->read($file);
  my @members = $zip->members();
  foreach (@members) {
    my $file=$_->fileName();
    if ($file=~m/.*\/(.*)$/) {
      $file = $1;
    }
    if ($file) {
      $zip->extractMemberWithoutPaths($_,$path."/".$file);
    }
  }
}
```

Thumbnails schreiben

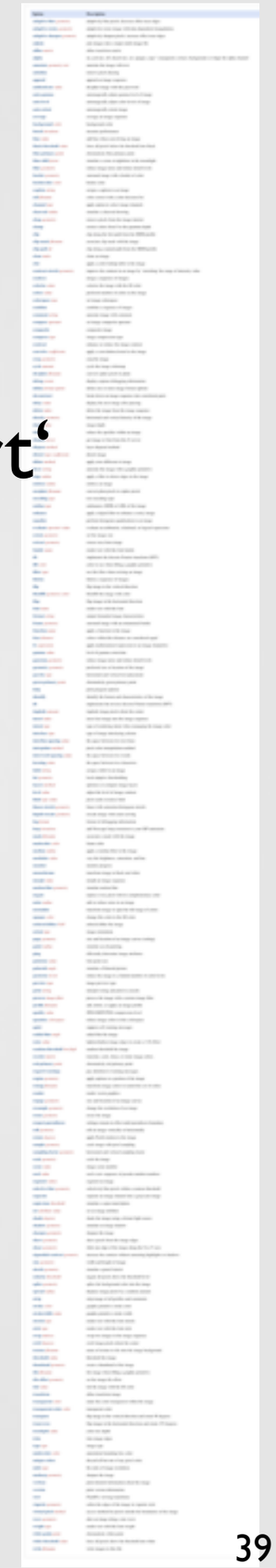
Thumbnails schreiben

- Kein Modul notwendig für Systemaufrufe
- system ruft Kommando auf
- Theoretisch auch sehr gutes ImageMagick-Modul für Perl

```
sub createThumbnails {
    my ($path) = @_;
    opendir(DIR, $path) or die "Can't open $dir: $!";
    my @allfiles = readdir(DIR);
    closedir(DIR);
    foreach $file (@allfiles) {
        if ($file =~ m/^\./) {
            unlink($path."/".$file);
            next;
        }
        next if ($file =~ m/^\./ || !($file =~ m/\.jpg$/) || $file =~ m/_T\.jpg$/);
        $inputfile = $path."/".$file;
        $file =~ /(.*)\.jpg$/;
        $outputfile = $path."/".$1."_T.jpg";
        $cmd = "/usr/local/bin/convert -resize 100x100 $inputfile $outputfile";
        system($cmd) || unlink($file);
    }
}
```

ImageMagick

- Kommandozeilen Bildverarbeitung mit umfangreichem Bildumwandlungstool ‚convert‘
- Dateiformate (> 100!)
  - Beispiele: AVI, BMP, JPEG, MPEG, PCX, PNG, PSD, SVG, TTF, WMF
- Optionen (s. rechts)
  - nur Einige: fill, rotate, resize, white-point
- Mehr Infos: www.imagemagick.org

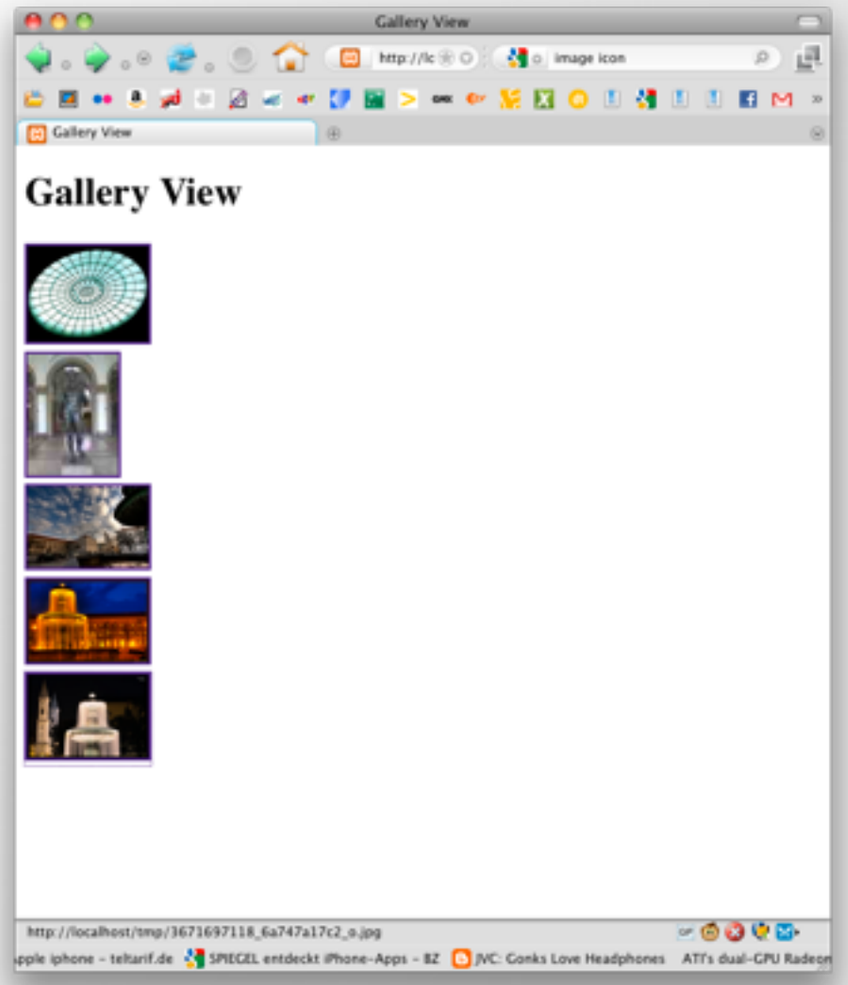


Gallery Seite erzeugen

Thumbnails schreiben

- Seite mit Template bauen und alle gefunden Bilder einfügen

```
my $UPLOAD_TEMPLATE = qq{
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<html><head><title>Gallery View (Perl)</title>
</head><body><h1>Gallery View (Perl)</h1>
};

sub generateWebsite {
       my ($path) = @_;
       my $html = $UPLOAD_TEMPLATE;
       opendir(DIR, $path) or die "Can't open $dir: $!";
       my @allfiles = readdir(DIR);
       closedir(DIR);
       foreach $file (@allfiles) {
              next if ($file =~ m/^\./ || !($file =~ m/\.jpg$/) || $file =~ m/_T\.jpg$/);
              $inputfile = $path."/".$file;
              $file =~ /(.*)\.jpg$/;
              $outputfile = $path."/".$1."_T.jpg";
              $html .= qq{<div class="image">};
              $html .= qq{<a href="$inputfile"><img src="$outputfile"/></a>};
              $html .= qq{</div>};
       }
       $html .= "Store file: $timeUpload seconds<br/>";
       $html .= "Unzip file: ".($timeUnzip-$timeUpload)." seconds<br/>";
       $html .= "Create thumbnails: ".($timeThumbnails-$timeUnzip)." seconds<br/>";
       $html .= "Overall: $timeThumbnails seconds<br/>";
       $html .= "</body></html>";
       return $html;
}

print $cgi->header();
print generateWebsite($UPLOAD_DIR);
```



Zeit messen

Zeitmessung

- Befehl time gibt die Unix-Zeit seit 1.1.1970 zurück.
- Lediglich sekundengenau.
- Time::HiRes liefert sehr genaue Zeit

```
use Time::HiRes qw (gettimeofday tv_interval);
my $timeStarted=gettimeofday;
# Heaving computing
my $timeTaken = gettimeofday-$timeStarted
print qq{Took $timeTaken seconds};
```

.pl Perl

Kompletter Code

Gallery Uploader Code

```
#!/usr/bin/perl
use utf8; # yes were using utf8 for everytihng
use CGI qw(:cgi);
use CGI::Carp qw(fatalsToBrowser);
BEGIN {
  # Stacktrace anzeigen. Auch für warnings
  $SIG{__DIE__} = sub { CGI::Carp::confess @_ };
  $SIG{__WARN__} = sub { CGI::Carp::confess @_; exit; };
}
use Time::HiRes qw (gettimeofday tv_interval);
use Archive::Zip qw( :ERROR_CODES :CONSTANTS );
my $timeStarted=gettimeofday; # note our start time in msecs# $start = Time.now()
our $cgi = new CGI;
my $timeUpload;
my $timeUnzip;
my $timeThumbnails;
our $UPLOAD_DIR = "./tmp";
our $HTML_TEMPLATE = qq{
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<html><head><title>File Upload (Perl)</title>
</head><body><h1>File Upload (Perl)</h1>
<form action="" method="post" enctype="multipart/form-data">
<input type="hidden" name="action" value="upload"/>
File name: <input name="file" type="file"/><br/>
<input name="submit" type="submit"/>
</form>
</body>
</html>
};
```

Gallery Uploader Code

```
my $UPLOAD_TEMPLATE = qq{
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<html><head><title>Gallery View (Perl)</title>
</head><body><h1>Gallery View (Perl)</h1>
};

sub errorPage {
    my ($text) = @_;
    print $cgi->header();
    print qq{<html><body><h1>$text</h1></body></html>};
    exit()
}

sub processFile {
    my ($form_field) = @_;
    $fileitem = $cgi->param($form_field);
    $filepath = $UPLOAD_DIR."/".$fileitem;
    open DAT,">$filepath" or die 'Error processing file: ',$!;
    # Dateien in den Binaer-Modus schalten
    binmode $filepath;
    binmode DAT;
    my $data;
    while(read $fileitem,$data,1024) {
      print DAT $data;
    }
    close DAT;
    return $filepath;
}
```

Gallery Uploader Code

```
sub unzip {
     my ($file, $path) = @_;
  $zip = Archive::Zip->new();
  $zip->read($file);
  my @members = $zip->members();
  foreach (@members) {
    my $file=$_->fileName();
    if ($file=~m/.*\/(.*)$/) {
      $file = $1;
    }
    if ($file) {
      $zip->extractMemberWithoutPaths($_,$path."/".$file);
    }
  }
}

sub createThumbnails {
     my ($path) = @_;
     opendir(DIR, $path) or die "Can't open $dir: $!";
     my @allfiles = readdir(DIR);
     closedir(DIR);
     foreach $file (@allfiles) {
          if ($file =~ m/^\./) {
               unlink($path."/".$file);
               next;
          }
          next if ($file =~ m/^\./ || !($file =~ m/\.jpg$/) || $file =~ m/_T\.jpg$/);
          $inputfile = $path."/".$file;
          $file =~ /(.*)\.jpg$/;
          $outputfile = $path."/".$1."_T.jpg";
          $cmd = "/usr/local/bin/convert -resize 100x100 $inputfile $outputfile";
          system($cmd) || unlink($file);
     }
}
```

Gallery Uploader Code

```
sub generateWebsite {
    my ($path) = @_;
    my $html = $UPLOAD_TEMPLATE;
    opendir(DIR, $path) or die "Can't open $dir: $!";
    my @allfiles = readdir(DIR);
    closedir(DIR);
    foreach $file (@allfiles) {
        next if ($file =~ m/^\./ || !($file =~ m/\.jpg$/) || $file =~ m/_T\.jpg$/);
        $inputfile = $path."/".$file;
        $file =~ /(.*)\.jpg$/;
        $outputfile = $path."/".$1."_T.jpg";
        $html .= qq{<div class="image">};
        $html .= qq{<a href="$inputfile"><img src="$outputfile"/></a>};
        $html .= qq{</div>};
    }
    $html .= "Store file: $timeUpload seconds<br/>";
    $html .= "Unzip file: ".($timeUnzip-$timeUpload)." seconds<br/>";
    $html .= "Create thumbnails: ".($timeThumbnails-$timeUnzip)." seconds<br/>";
    $html .= "Overall: $timeThumbnails seconds<br/>";
    $html .= "</body></html>";
    return $html;
}
```

Gallery Uploader Code

```
if ($#ARGV >= 1) {
    # wir haben ein command line argument!
    print "Command line\n";
    my $filepath = $ARGV[0];
    $timeUpload = gettimeofday-$timeStarted;
    unzip($filepath, $UPLOAD_DIR);
    $timeUnzip = gettimeofday-$timeStarted;
    createThumbnails($UPLOAD_DIR);
    $timeThumbnails = gettimeofday-$timeStarted;
    my $html = generateWebsite($UPLOAD_DIR);
    open (MYFILE, '>gallery.html');
    print MYFILE $html;
    close MYFILE;
    exit()
} else {
    if (defined $cgi->param('action') && $cgi->param('action') eq "upload") {
        my $filepath = processFile('file');
        $timeUpload = gettimeofday-$timeStarted;
        unzip($filepath, $UPLOAD_DIR);
        $timeUnzip = gettimeofday-$timeStarted;
        createThumbnails($UPLOAD_DIR);
        $timeThumbnails = gettimeofday-$timeStarted;
        print $cgi->header();
        print generateWebsite($UPLOAD_DIR);
        exit()
    }

    print $cgi->header();
    print $HTML_TEMPLATE;
}
```

Nächste Woche: PHP

Literatur

- Perl garbage collector and swap - http://www.perlmonks.org/?node_id=337359
- Perl - Wikipedia - http://en.wikipedia.org/wiki/Perl
- Perl Hash Howto - http://www.cs.mcgill.ca/~abatko/computers/programming/perl/howto/hash/
- Perl - $_ and @_ - http://www.wellho.net/mouth/969_Perl-and-.html
- Creating (and maintaining) Perl modules - http://mathforum.org/~ken/perl_modules.html

Bildnachweis

- http://www.creativeuncut.com/gallery-07/art/mlpit-mario-baby-hammer.jpg
- http://www.dotolearn.com/picturecards/images/imageschedule/proud_1.gif
- http://www.hakstpoelten.ac.at/buchoase/Buch.gif
- http://school.discoveryeducation.com/clipart/images/arteasel4c.gif
- http://www.gg-schnitt.at/wp-content/uploads/2009/01/lego_brick.png
- http://www.helliot.com/cms/images/stories/logo/logo_school.gif
- http://www.clker.com/clipart-window-icon.html
- http://globaleuropeans.com/uploads/images/GE_global%20projects%202.jpg
- http://www.sbac.edu/~tpl/clipart/Animals%20and%20Insects/bug%20cartoon%2002.jpg
- http://jasoncirillo.files.wordpress.com/2009/03/pong.jpg