Perl

Aktuelle Programmiersprachen
WS 06/07
MIB :: Axel Reusch
ar047@hdm-stuttgart.de

Inhalt

- Allgemein
- Einsatz
- Sprachkonzepte
- OO-Prinzipien
- Fehlerbehandlung
- Perl vs. Java
- Entwicklungsumgebung
- Unterstützung Teamentwicklung
- Programmbeispiele

Grundlegende Infos

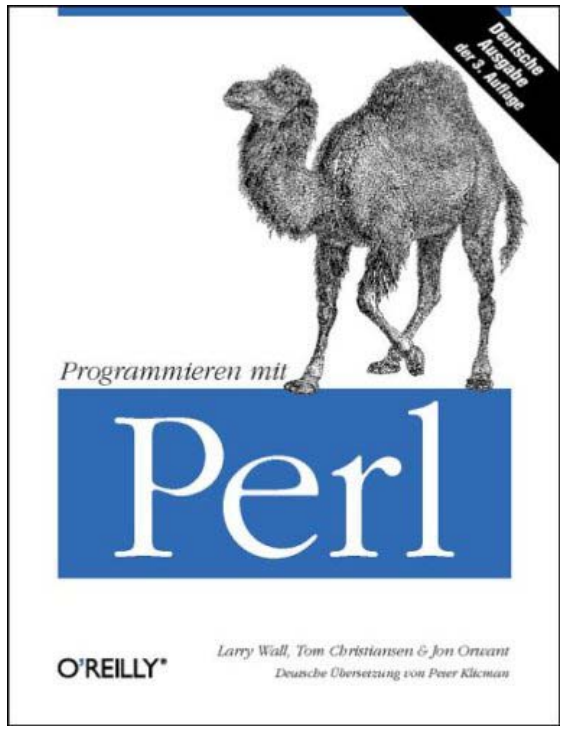
- Scriptsprache
- plattformunabhängig
- interpretiert

- entwickelt 1987 von Linguist Larry Wall (tätig als NW-Admin)
- „practical extraction and report language"
- „pathologically eclectic rubbish lister"

- Konzepte: C, UNIX (Shell) und andere Einflüsse
  - siehe Kapitel „Konzepte"
- damals: System- & Netzwerkadministration
- Umbenennung: Pearl -> Perl

- Aktuell Version 5.8.8 (Jan. 2006)

Grundlegende Infos



- Perl: Programmiersprache
- perl: Interpreter

- DIE Perl-Bibel: „Programmieren mit Perl"
  - „Das Kamel-Buch"
  - Zusammenhang
    - Perl <-> Dromedar: KEINER

- Offizielle Perl-Seite
  - http://www.perl.org

- ActivePerl frei erhältlich unter http://www.activestate.com

- CPAN Modulsammlung
  - Comprehensive Perl Archive Network http://www.cpan.org

Einsatz

- Zu Beginn UNIX-Werkzeug
  - Verarbeitung Textdateien
  - Steuerung anderer Programme
  - Ausgabe von Berichten
  - Skripte Verbindung zw. Inkompatibler Software
- Nutzung mit Ausbreitung www - cgi
  - Webserver, Datenbanken, Programme und Daten verbinden
    - Erzeugung dynamischer HTML-Dokumente (siehe PHP)
    - Programme zB zum Spamschutz (SpamAssasin), E-Mail (Open Webmail), Trouble Ticket Systeme…

Einsatz

- PHP ist moderner, also wer nutzt noch Perl???
  - viele große und kleine Seiten und Internetdienste
    - *MovableType*
    - *LiveJournal*
    - *SlashDot.org*

amazon.com

Konzepte

- „There's more than one way to do it"
  - Freiheit des Programmierers (|| = or &&=and)
- „Perl makes easy jobs easy and hard jobs possible"
  - Effektivität

Einflüsse:

- C
- Unix
- Ruby, …

- Kontext-sensitiv
- Dynamische Variablentypzuordnung
- als eine der ersten Sprachen: Reguläre Ausdrücke
  - schnelle Implementierung
- Ab Version 5: Objektorientierung

Konzepte

- Basisdatentypen
  - Skalare Datentypen
  - Arrays
  - Hashes (assoziative Arrays)

$ für Skalare:
@ für Arrays:
% für Hashes:
& für Funktionen:

```
$scalar = 5;
@array = (“Tom”, “Jones”);
%hash = ("Er","Porsche","Du","Trabbi"); # ->
&function(Blub,Titanic);  #Aufruf
```

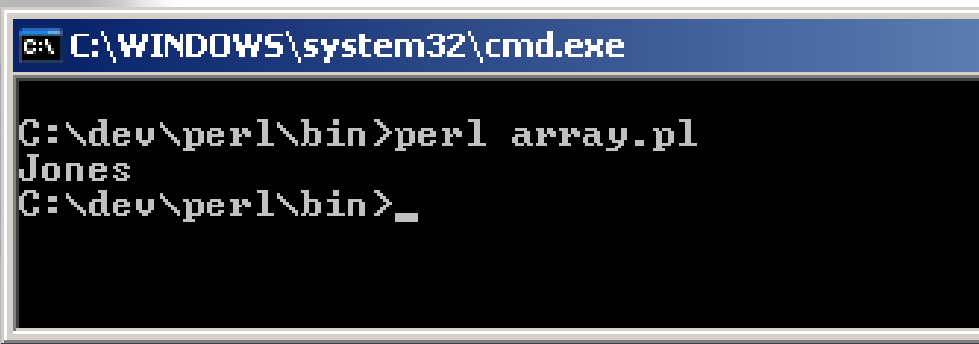
- Variablenkennzeichnung durch Prefix

Konzepte

- Basisdatentypen
  - Skalare Datentypen
  - Arrays
  - Hashes (assoziative Arrays)

$ für Skalare:	$scalar = 5;
@ für Arrays:	@array = (“Tom”, “Jones”);
% für Hashes:	%hash = ("Er","Porsche","Du","Trabbi"); # ->
& für Funktionen:	&function(Blub,Titanic); #Aufruf

- Für Ausgabe
  - print $array[1];

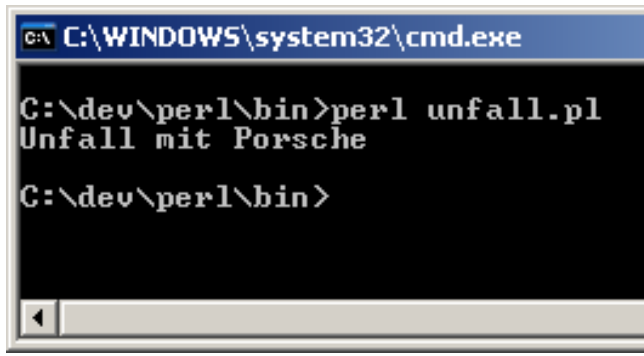


Konzepte

- Basisdatentypen
  - Skalare Datentypen
  - Arrays
  - Hashes (assoziative Arrays)

$ für Skalare:	$scalar = 5;
@ für Arrays:	@array = (“Tom”, “Jones”);
% für Hashes:	%hash = ("Er","Porsche","Du","Trabbi"); # ->
& für Funktionen:	&function(Blub,Titanic); #Aufruf

- Für Ausgabe
  - print „Unfall mit $hash{Er}\n";



Konzepte

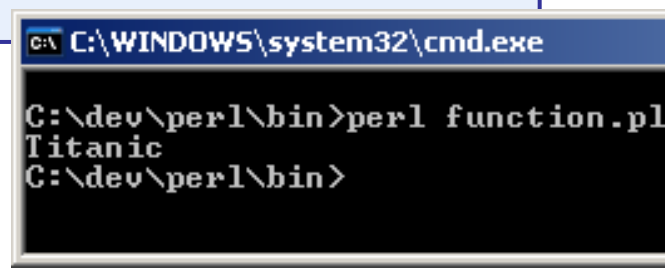
- Basisdatentypen
  - Skalare Datentypen
  - Arrays
  - Hashes (assoziative Arrays)

$ für Skalare:
@ für Arrays:
% für Hashes:
& für Funktionen:

```
$scalar = 5;
@array = (“Tom”, “Jones”);
%hash = ("Er","Porsche","Du","Trabbi"); # ->
&function(Blub,Titanic);  #Fkt.Aufruf
```

- Parameterübergabe über lokales Array @_
- Bsp-Aufruf in Fkt:

```
sub function{
        print $_[1];
}
```



Konzepte

- Kontrollstrukturen
  - Vergleichbar mit C, Java, JavaScript
- Bedingte Verarbeitung

```
if (<Bedingung>) {<Anweisungen>}
 [elsif (<Bedingung>) {<Anweisungen>}]
 [else {<Anweisungen>}]
```

```
unless (<Bedingung>) {<Anweisungen>}
[else {<Anweisungen>}]
```

If-Shorty:

```
<Bedingung> ? <Anweisung 1> : <Anweisung 2>;
```

Konzepte

- Schleifen

```
[label:] while (<Bedingung>) {<Anweisungen>}
[continue {<Anweisungen>}]

[label:] until (<Bedingung>) {<Anweisungen>}
[continue {<Anweisungen>}]

[label:] for ([<Startanweisung>];
[<Bedingung>];[<Updateanweisung>]) {<Anweisungen>}
[continue {<Anweisungen>}]

[label:] for[each] [[my] $element] (<Liste>) {<Anweisungen>}
[continue {<Anweisungen>}]
```

Konzepte

- Schleifen

```
do {<Anweisungen>} while <Bedingung>;

do {<Anweisungen>} until <Bedingung>;
   # mindestens eine Ausführung
```

- Kein Case- oder Switch in Perl 5!

Konzepte

- Subroutinen / Funktionen

```
sub tolleRoutine {
    <Anweisungen>
    return blub;
}
```

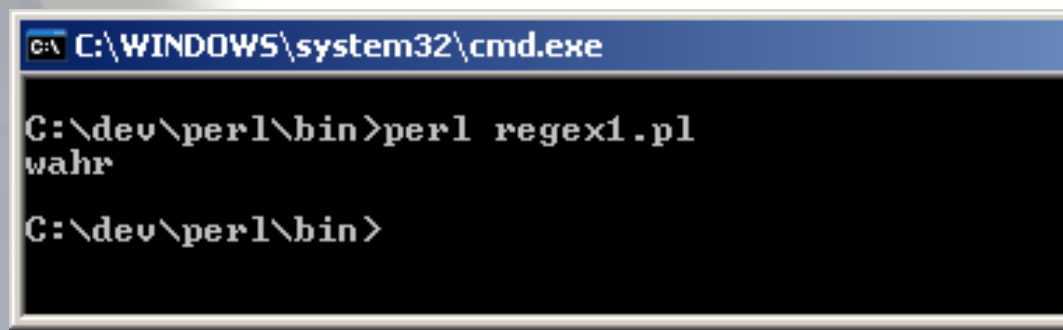
- Entsprechend Funktionen (siehe vorher)
  - @_ als Parameterübergabe

Konzepte

- Reguläre Ausdrücke – einfaches Beispiel

```
#!/usr/local/bin/perl -w
use strict;
my $t="demo";
if($t =~ /e/){ print "wahr\n"} else {print ("falsch\n")};
```

- Suche nach einzelnem Buchstaben in String



Konzepte

- Reguläre Ausdrücke – einfaches Beispiel

```
#!/usr/local/bin/perl -w
use strict;
my $t="demo";
if($t =~ /e/){ print "wahr\n"} else {print ("falsch\n")};
```

- =~ veranlaßt Suche (kontext-sens.) nach Zeichen e im String $t
- // umrahmt regulären Ausdruck
- Bei mehreren Zeichen entsprechend:
  - ($t =~ /emo/)
  - ($t =~ /e.o/) # „.“ als Joker

Konzepte

- Reguläre Ausdrücke – einfaches Beispiel

```
#!/usr/local/bin/perl -w
use strict;
my $t="demo";
if($t =~ /e/){ print "wahr\n"} else {print ("falsch\n")};
```

- Suche nach Zeichengruppen (Zeichenklassen) mit []
  - $t =~ /[aeiou]/ #kommt Vokal vor?
  - [^aeiou] entspricht KEINE Vokale
  - [a-z] ist Bereich

Konzepte

- Reguläre Ausdrücke: Kurzformen Zeichengruppen

Zeichen	**Entsprechung**	**Bedeutung**
\d	[0-9]	Ziffer
\D	[^0-9]	Gegenstück zu \d
\w	[a-zA-Z_0-9]	Buchstabe, Unterstrich oder Ziffer
\W	[^a-zA-Z_0-9]	Gegenstück zu \w
\s	[\t\n\f\r]	Leerzeichen
\S	[^ \t\n\f\r]	Gegenstück zu \s

Konzepte

- Reguläre Ausdrücke – einfaches Beispiel

```
#!/usr/local/bin/perl -w
use strict;
my $t="demo";
if($t =~ /e/){ print "wahr\n"} else {print ("falsch\n")};
```

- Suche nach Zeichen die mehrfach vorkommen
  - $t =~ /e{1,3}/ #kommt e 1,2 od. 3 mal vor
- Suche nach mehreren Zeichen die mehrfach vorkommen ()
  - $t =~ /(em){1,3}/ #kommt em 1,2 od. 3 mal vor

Konzepte

- Reguläre Ausdrücke: Kurzformen Zeichengruppen

Quantifizierer	**Bedeutung**
{*n*,*m*}?	mindestens *n*-mal, höchstens *m*-mal
{*n*}?	genau *n*-mal (äquivalent zu {*n*})
{*n*,}?	mindestens *n*-mal
??	höchstens einmal
+?	mindestens einmal
*?	beliebig oft

- Es gäbe noch vieles mehr zu sagen…

Objektorientierung

- OO ab Perl 5
  - Syntax aus vorhandenen Sprachelementen
- Modularisierung
  - Auslagerung in externe Module (*.pm) (können Klassen sein!)
  - Notwendig für OO sind Packages (siehe Java)
- Kapselung - Gültigkeitsbeschränkung - Sichtbarkeit
  - my (wie private)
    - my $gurkenhobel=5;
  - local (wie my, aber sichtbar für Subroutinen)

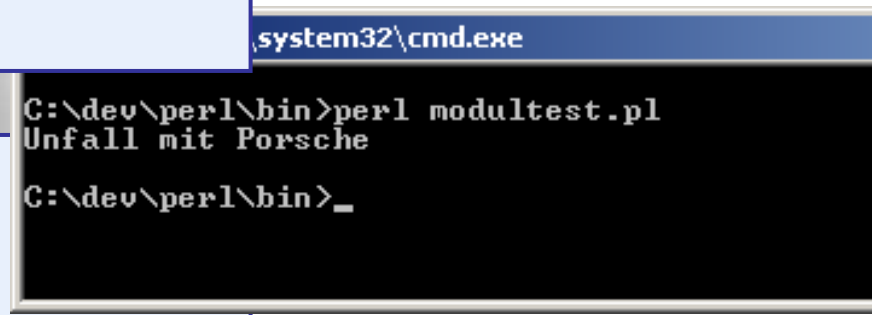
Objektorientierung

- Modularisierung Grundprinzip
  - Funktionen und Variablen werden ausgelagert
  - Nutzung im Programm durch use
  - Bsp. gurkenhobel.pm

```
package gurkenhobel; #für Modul unerheblich
%hash = ("Er","Porsche","Du","Trabbi");
print "Unfall mit $hash{Er}\n";
```

  - Aufruf in modultest.pl

```
#! /usr/bin/perl –w
use strict;
use gurkenhobel;
```



Objektorientierung

- Modularisierung Grundprinzip
  - Paketdeklaration mit package
  - use + Modulname
  - Bsp. gurkenhobel.pm

```
package gurkenhobel; #für Modul unerheblich
%hash = ("Er","Porsche","Du","Trabbi");
print "Unfall mit $hash{Er}\n";
```

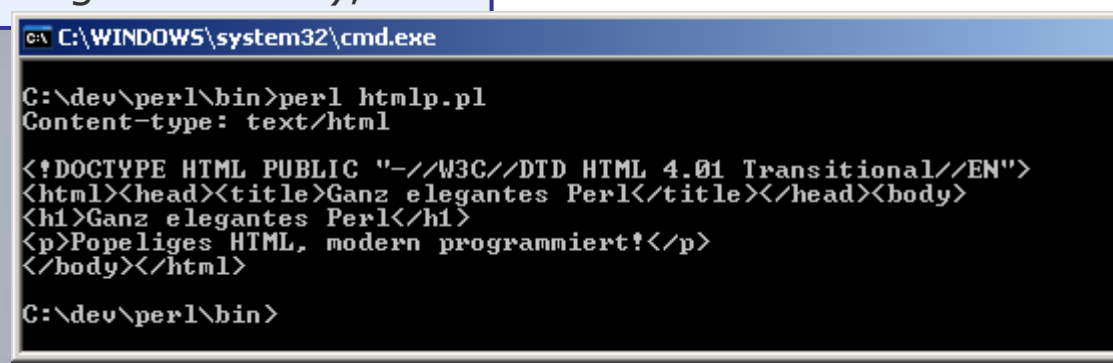
  - Aufruf in modultest.pl

```
#! /usr/bin/perl –w
use strict;
use gurkenhobel;
```

Objektorientierung

- Modularisierung als Klasse (Bsp. aus selfhtml)
  - Htmlp.pl

```
#!/usr/bin/perl -w
    use strict;
    use CGI::Carp qw(fatalsToBrowser);
    use htmlprint;
    my $html = htmlprint -> new();
    $html->Anfang("Ganz elegantes Perl"); …
```



Objektorientierung

- Modularisierung als Klasse
  - Htmlp.pl

```
#!/usr/bin/perl -w
    use strict;
    use CGI::Carp qw(fatalsToBrowser);
    use htmlprint;
    my $html = htmlprint -> new();
    $html->Anfang("Ganz elegantes Perl"); …
```

- use cgi für cgi-Nutzung (ansprechen von Modul cgi.pm, opt. Param.)
- new() Konstruktor
  - Nicht Teil von Perl! (kann auch anders heissen!)
  - Ansprechen der Methode new() in htmlprint.pm

Objektorientierung

- Modularisierung als Klasse
  - Htmlprint.pm

```
package htmlprint; #identisch mit Modulname
sub new { my $Objekt = shift;
my $Referenz = {}; bless($Referenz,$Objekt); return($Referenz);}
sub Anfang {
  my $Objekt = shift;  my $Titel = shift;
  … # Teile entnommen
  print "<html><head><title>$Titel</title></head><body>\n"; …
}
```

Objektorientierung

- Modularisierung als Klasse
  - Htmlprint.pm

```
package htmlprint; #identisch mit Modulname
sub new { my $Objekt = shift;
my $Referenz = {}; bless($Referenz,$Objekt); return($Referenz);}
sub Anfang {
  my $Objekt = shift;  my $Titel = shift;
  … # Teile entnommen
  print "<html><head><title>$Titel</title></head><body>\n"; …
}
```

Objektorientierung

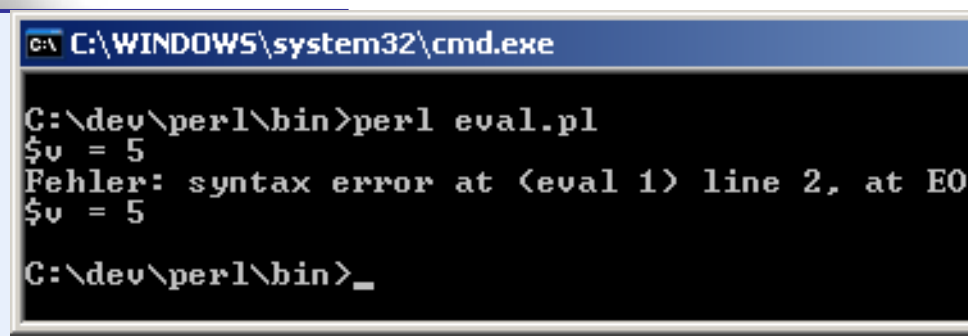
- Modularisierung als Klasse
  - Htmlprint.pm

```
sub new {
… my $Objekt = shift;
                        #Zuweisung erstes Objet New-Parameter
                        #autom. Klassenname (htmlprint)
… my $Referenz = {};
                        # Referenz für bless()
… bless($Referenz,$Objekt); …}
                        # Verbindung zw. Referenz und Objekt
sub Anfang {…}
                        #Methode in Klasse/im Objekt
```

Fehlerhandling

- Use strict; um Variablendeklaration zu erzwingen
- Wichtig:
  - -w zur Warnungsausgabe
- Konzept ähnlich Try/Catch aus Java: eval

```
#!/usr/local/bin/perl -w
use strict;
my $v = 5;
print "\$v = $v\n";
eval '$v+';      # FEHLER !
if($@) { print "Fehler: $@" }
print "\$v = $v\n";
```



Fehlerhandling

- $@ internes Array das erzeugt wird wenn Fehler auftaucht

```
#!/usr/local/bin/perl -w
use strict;
my $v = 5;
print "\$v = $v\n";
eval '$v+';
if($@) { print "Fehler: $@" }
print "\$v = $v\n";
```

Perl vs Java

- OO (ab Perl 5)
- Perl eher für kleine & mittlere Projekte geeignet
- Perl teilweise unausgereift (siehe Threads)
- Langsame Entwicklung

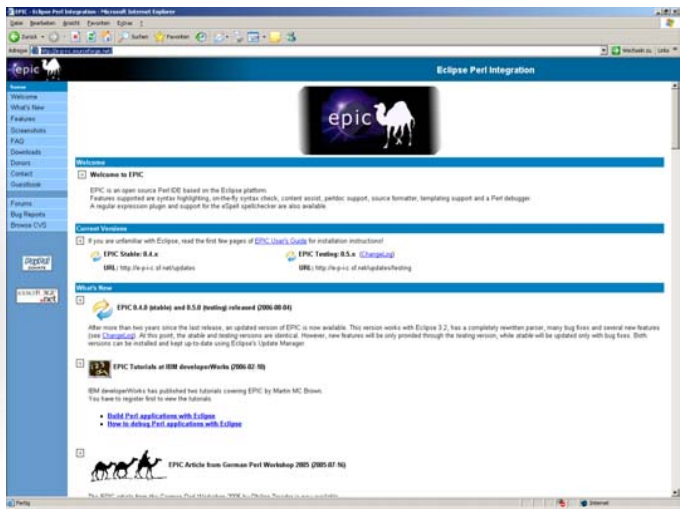
- OO (dafür konzipiert)
- Projekte in jeder Größe verwirklichbar
- Ausgereift

- Aktuell

Verschiedene Einsatzgebiete

Entwicklungsumgebung

- Editoren des OS
  - z.B. Notepad
- Viele Editoren erhältlich über www (Freeware)
  - Über Google-Suche
- Eclipse-PlugIn EPIC
  - http://e-p-i-c.sourceforge.net



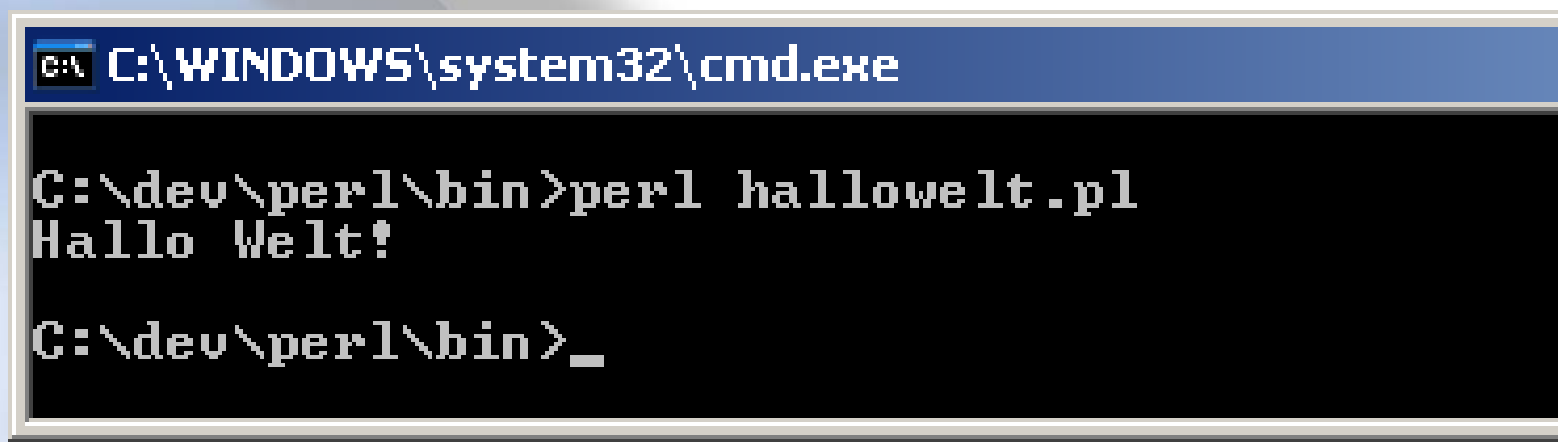
Teamentwicklung

- Kein direkter Support
- Möglichkeit vorhanden durch Module und Ausführung externer Perl-Dateien über zB
  - require hulla.pl;
- Objektorientierung

Beispiele

- Das legendäre „Hallo Welt"-Programm:

```
#! /usr/bin/perl –w

print („Hallo Welt!\n");
```



Beispiele

- Das legendäre „Hallo Welt"-Programm:

```
#! /usr/bin/perl –w

print („Hallo Welt!\n");
```

- Identifikation Perl-Programm
- # normalerweise Kommentar
- Optionales Argument „–w":
  - Standard
  - Zusätzliche Warnung bei bei evtl. gefährlichen Konstrukten

Beispiele

- Das legendäre „Hallo Welt"-Programm:

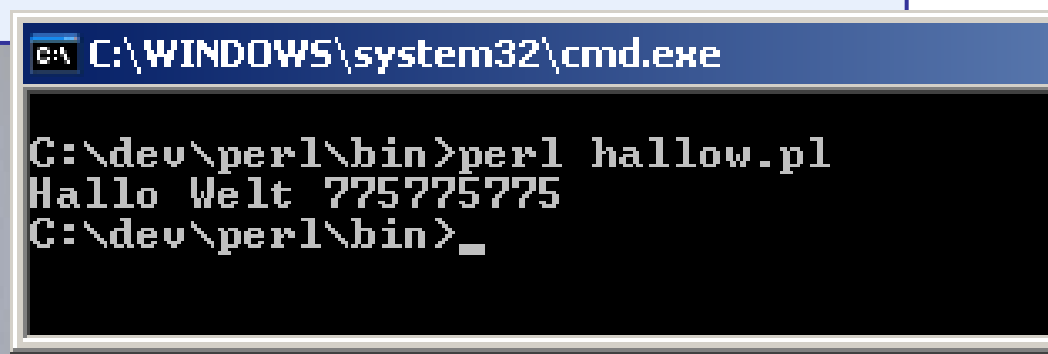
```
#! /usr/bin/perl –w

print („Hallo Welt!\n");
```

- ähnlich C
  - „Hallo Welt" Zeichenkette
  - „\n" Newline

Beispiele

- Variablen / Stringaddition

```
#! /usr/bin/perl –w
$var=(„Hallo “);
$Var=775;
$zahl=$Var;
$Var=(„Welt “);
print $var.$Var.$zahl x 3;
```



Beispiele

- Variablen / Stringaddition

```
#! /usr/bin/perl –w
$var=(„Hallo “);
$Var=775;
$zahl=$Var;
$Var=(„Welt “);
print $var.$Var.$zahl x 3;
```

- „Case-sensitive“
- Variablen müssen nicht deklariert werden
- „.“ für String-Addition
- „x“ für String-Multiplikation

- Pro ☺
  - Schnell & effektiv
  - gute Community (Support)
  - freie Verfügbarkeit

Fazit

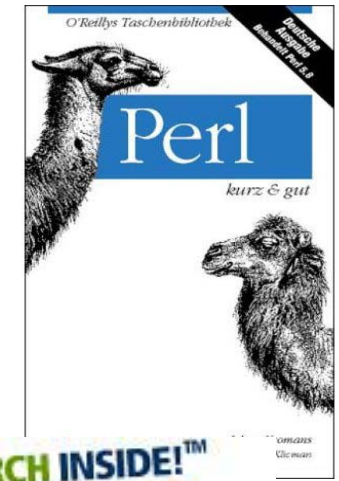
- Contra ☹
  - Wirkt oftmals kryptisch
  - PHP schlägt Perl
  - Langsame Entwicklung (Perl 6)
    - Kein Sponsoring

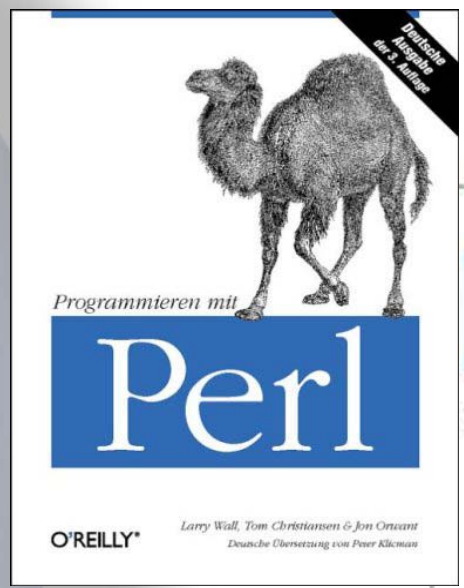
Literatur/Link-Tips

(blau eingefärbt = Quelle)

Bücher

- Programmieren in Perl (Larry Wall, Randal L. Schwarz, Tom Christian)
  - O'Reilly
  - Ca. 56,- Euro
- Einführung in Perl (Randal L. Schwarz, [Tom Christian])
  - O'Reilly
  - Ca. 34,- Euro
- Perl – Kurz und gut (Johan Vromans) - NICHT FÜR ANFÄNGER
  - O'Reilly
  - Ca. 8,- Euro





Web

- Perlunity.de
- Perl.org
- Cpan.org
- http://de.selfhtml.org/perl/
- Wikipedia.org
  - Mehr Links !!!
- perl-seiten.privat.t-online.de

Vielen Dank für Ihre Aufmerksamkeit!

Fragen?

Aktuelle Programmiersprachen
WS 06/07
MIB :: Axel Reusch
ar047@hdm-stuttgart.de