The Perl programming language

Introduction	**(1 lesson)**
Part 1 Basic Concepts in Perl	**(4 lessons)**
Part 2 Object orientation basics	**(2 lessons)**
Part 3 Working with perl modules	**(4 lessons)**

Wolfgang Friebel
DESY Zeuthen

Introduction

Perl resources

Data types

The Camel is commonly associated with Perl
(displayed on the cover of the O‘Reilly book “Programming Perl“)

Perl Slogans

- The three big virtues of a programmer
  laziness, impatience and hubris
- There is more than one way to do it (TMTOWTDI)
  Comparison of natural languages and Perl
- Yet another perl hacker (CPAN/misc/japh)
  many scripts to produce this slogan
- Far more than you ever wanted to know series
  articles by T. Christiansen (e.g. on www.perl.com)

Contents of the lectures

- **Part 1 Basic Concepts in Perl**
  - ▲ Reading and writing your data
  - ▲ processing the data: regular expressions
  - ▲ processing the data: functions and modules
  - ▲ tuning, debugging and documenting your scripts
- **Part 2 Object orientation basics**
- **Part 3 Working with perl modules**

Perl Resources

- Books
- Journals, Web Sites, Newsgroups
- Talks, Tutorials, Papers
- Perl Software
- Perl at your Fingertips (online manuals)
- The Perl Installation at DESY

Books

- **Reference and Recipes**

  Programming Perl by L.Wall, T.Christiansen, R.Schwartz, 3rd edition 2000 (covers perl 5.6.0) (Camel book)

  Perl Cookbook by T.Christiansen and N.Torkington (Ram book)

  Effective Perl Programming by J.N.Hall and R.Schwartz

- **Beginner**

  Learning Perl and Learning Perl on Win32 Systems by R.Schwartz and T.Christiansen (Llama book)

- **Advanced**

  Advanced Perl Programming by S.Srinivasan (Panther book)

  Object Oriented Perl by D.Conway

Books (2)

- **Topical books**

  Mastering Regular Expressions by J. Friedl

  Mastering Perl/Tk by S.O.Lidie/N.Walsh

  How to Set Up and Maintain a Web Site by L.Stein

  Programming the Perl DBI by A.Descartes and T.Bunce

  Perl for System Administration by D.N.Blank-Edelman

- **Quick Refs**

  perl 5.8 (Vromans)

  perl/Tk Version 8 (Lidie)

- **Many more books of bad and good quality**

  see e.g. http://www.perl.com

Journals, Web Sites, Newsgroups

- Journals
  - The Perl Journal (TPJ) (for contents see www.tpj.com)
  - Articles in Linux Journal, Webtechniques etc.
- Web sites
  - www.perl.com - **the** primary address for perl
  - CPAN (Comprehensive Perl Archive Network)
    ftp://ftp.uni-hamburg.de/pub/soft/lang/perl/CPAN
  - www.northbound-train.com/perlwin32.html - for Win32
- Newsgroups
  - comp.lang.perl.*

Talks, Tutorials, Papers

- Online Documents

  (small selection from www.perl.com)

  - Perl5 by Example (M. Medinet)
  - Advanced topics in perl programming (Christiansen)
  - Practical Web Site Maintenance with Perl (Klein)
  - Perl Quick reference guide (for perl 5.004)
  - Perl/Tk Pocket reference (for perl/Tk 4)
  - Perl für UNIX und C-Kenner (F. Hajji, Köln 1997)
- some docs at www.desy.de/zeuthen/~friebel/perl

Perl Software

- first place to look at is CPAN (http, ftp, see above)
- nearest site: Hamburg, many mirrors
- source code from most examples in books available online

Perl at your Fingertips

- Version (`perl -v`), Installation details (`perl -V`)

  Latest version 5.8.1, standard at DESY 5.8.0

  UNIX and NT versions from same source tree

- Online Information with man and perldoc
  - perldoc more flexible than man
  - `perldoc perl` is equivalent to `man perl`
  - `perldoc -f function` shows function definition
  - `perldoc perllocal` lists installed noncore modules
  - `perldoc -q String` looks for string in “FAQ’s”
  - `perldoc DBD::mysql` displays module description
  - See also switches `-h, -l, -m`

The Perl Installation at DESY (UNIX)

- /usr/bin/perl and /usr/local/bin/perl do usually differ only /usr/local/bin/perl which is a link to the (/opt)/products/perl tree is maintained by me
- /usr/bin/perl comes with the UNIX system
- Current version is 5.8.0
- Older versions (5.005, 5.6.0, 5.6.1) from /products/perl/*version*/bin/perl

  make perl 5.6.0 the current version with

  `ini perl56`, switch back with `ini -d perl56`

The Perl Installation at DESY (NT)

- use netinstall package for version 5.005
  It contains the ActiveState port (Build 522)
- Newer versions from http://www.activestate.com
  - ▲ Also on ftp://ftp.ifh.de/pub/windows/perl/
  - ▲ Plans to upgrade perl on Windows do exist
- For program development editors with command line editing, syntax highlighting and history are recommended (e.g. emacs, xemacs, vim)
- Integrated Development Environments are also existing (PerlBuilder, CodeMagic and others)

Command Line Example

- `perl -e 'print ”hello world!\n"'` (UNIX)
- `perl -e "print qq(hello world!\n)"` (NT)

  The -e flag requires a valid perl program as argument

  Variable substitution and special character handling like \n only within double quotes, not within single quotes

  The Windows Shell does not properly treat single quotes

  qq is a function (generalized double quotes)

  print is also a function (does not automatically emit CR/LF)

Introductory Script

Important constructs

- File intro.pl:

```
#!/usr/local/bin/perl -w    #-w is your friend
#Our first example
print "What is your name?\n";
$who = <STDIN>;    #read a line from STDIN
chomp $who;        #see also perldoc -f chomp
print "Welcome on a $^O system, \u$who.\n"
```

Beginners can ignore that

Execution by `perl intro.pl`

or on UNIX simply by `intro.pl` if file is executable

Basic Syntax Rules

- Flags on the first line are also respected under NT
- Characters after a # up to EOL are comments
- Commands are separated by ;
- White Space is significant only in strings
- Simple variables start with $, Variables with punctuation characters are Perl Special Variables

Data Types

- 3 fundamental data formats (plus pointers)

  Strings, Integer, Double Precision Numbers (and Pointers)

- Data types are built from these data formats:

Constants	123 “a”	References	\$abc
Variables	$abc	File Handles	STDIN
Lists	(1, 2, $abc)	Formats	HEADER
Arrays	@abc	Objects	$obj
Hashes	%abc	Globs	*glob

Numbers

- no surprises (except maybe _)
  - `123, 0755, 0x1f, 3e10, 1.5e-6, 10_000`

    dec oct hex float
  - Operations on numbers `+ - * / % ** ++ --`
  - Comparisons `< <= > >= == != <=>`
  - Bit Operations & | ^ ~ << >>

Strings

- literal Strings: `'Value: $100'`
- Strings with Variable and Backslash Interpretation: `"Welcome $who\n\\061 is octal \061\n";`
- Backslashes (\) in interpreted Strings written as \\
  - \ and / are interchangeable in path specifications for the open function (but **not** e.g. for unlink),

```
open F "C:/perl/docs";      # the same as
open F "C:\\perl\\docs";    # this statement
```

String Operations

- Concatenation operator `.` (Dot) `'abc'.'def'`
- Repetition operator **`x`** `'#' x 70`
- String Comparisons: **`lt le gt ge eq ne cmp`**
  - **`$a cmp $b`**
  - **`yields -1 if $a lt $b`**
  - **`yields  0 if $a eq $b`**
  - **`yields  1 if $a gt $b`**

Long Strings (Here Documents)

- Long Strings with Here Documents:

```
$long = <<"EOF";                    #or <<EOF;
Hello, $who,
Please try also 'EOF' and `EOF`.
EOF
```

- No Space between << and EOF, don‘t forget ; !!!
- No other chars than EOF on last line !!!
- The `'EOF'` Notation takes the text literally
- The `` `EOF` `` Notation interprets all lines as shell commands, which get replaced by its output

Sample commands with Strings

```
print "=" x 70, "\n";


print <<END_OF_TEXT;
This is the $^O operating system
END_OF_TEXT


print <<`EOF2`;
dir
EOF2
```

Quotes

- **Customary**

```
'text'   #no Interpolation
"$a\n"   #Interpolation
`dir`    #quoted execution

/match/#Pattern (Interpol.)
```

- **Generic**

```
q[]     #any delimiter
qq{}    #matching braces,...
qx()    #
qw()    #quote words
m//     #pattern match
qr()    #regexp (since 5.005)
```

The method of choice is mostly a question of readability
Generic quotes allow nesting

Lists

- Lists consist of 0..n comma delimited words enclosed in parentheses

```
(1, 2, 3, "abc", 3.14)
```

quotes optional without `use strict;` not recommended

String or function name

- List elements can be addressed:

```
(1, 2, 3, abc, 3.14)[1,3]
```

- Lists may be assigned to (no constants on LHS!):

```
($a,$b) = ($b,$a);   # flip elements
```

- many functions operate on lists

Variables

- no declaration required, initial value is **`undef`**
- declaration should be enforced by **`use strict;`**
  - ▲ Typos will be catched (uninitialized data with **`-w`**)
- Scalar Variables

  start with **`$`**, followed by the name **`$what`**

  the name can contain _ and 0..9 **`$a_01`**

  the name can be an expression, delimited by **`{}`**

  Example: **`${"wh"."at"}`** is the same as **`$what`**

Special (scalar magic) Variables

- have a single non alphanum character as name
- most important magic Variable is `$_`
  - also known as default argument
  - implicitly used in many functions, if no arg given
  - default iterator variable in loops
  - pattern matches/substitutions work by default on **`$_`**
  - with **`<FH>`** input records can be placed in **`$_`**
- have a long name if `use English;` was given
- many other useful special vars, e.g `$^O, $^T, $^0`

Array Variables

- Array Variables

  Array names are preceded by a @ character

  ```
  @name = ("Friebel", "W.");#use notation below
  @name = qw(Friebel W.);   #less punctuation chars
  ```

  Element count starts at Zero (more precisely at `$[`)

  Last element of `@name` is `$#name` (-1 if empty)

  Assigning to `$#name` changes array length !!!

- Elements of Array Variables

  preceded with `$`, not with `@`, array subscripts are enclosed in `[]`

  negative subscripts count from end of array

  ```
  $fam = $name[0]; $initial = $name[-1];
  ```

Special (magic) Arrays

- name composed of capital Letters only
- user defined Array names should not use all capital Letters
- Most important special Arrays are **`@ARGV`**, **`@_`** and **`@INC`**
  - **`@ARGV`** and **`@_`** are used for passing of Arguments
  - **`@INC`** contains the search path for Perl Modules

Hashes

- Hashes are associative Arrays
- Hash names are preceded by a % character
- Addressing of elements is done by strings (**key**) as opposed to numbers in the case of arrays
- Hashes can be initialized using a list

```
%name = (´initial´, ´W.´, ´family´, ´Friebel´);
```

  - ▲ this is written more clearly using the arrow notation with key/value pairs
  - ▲ then the quotes around the keys can be omitted

```
%name = ( initial => ´W.´,
          family  => ´Friebel´);
```

Hash Elements

- Elements of a hash are
  - preceded with **`$`**, not with **`%`**, hash subscript (key) enclosed in **`{}`**
  - Quotes around strings used as hash key may be omitted
  - **`$fname = $name{family};`**
- Access to **all** Elements of Hashes
  - provided through the functions **`each, keys, values`**
  - **`each`** returns a key/value pair on each invocation
  - pairs are returned in an apparently random order

Hash Elements (2)

- Test for the existence of a key/value

  is achieved by the functions **`exists`** and **`defined`**

```
print ”key exists " if exists $hash{key};
print ”value defined " if defined $hash{key};
print "\n";
```

- To erase a hash value assign **`undef`** to the element

```
$hash{key} = undef;
```

- To erase a key/value pair, **`delete`** it

```
delete $hash{key};
```

Special (magic) Hashes

- name composed of capital Letters only
- Most important special Hashes are **%ENV** and **%SIG**
  - **%ENV** contains the Environment Variables of the process
  - can be read out and assigned to
  - Assignment may change system properties

```
$ENV{TEMP}= ´C:/TEMP´;#System interaction
$ENV{LD_LIBRARY_PATH} = ´/usr/local/lib´;
```

  - **%SIG** is responsible for system interrupts

Magic Variables Demo

```
# $$ contains the process ID, used to create a temporary file name
$tmpfile = "\\temp\\perlwf.$$"; #System dependent and insecure
# For a better solution see perldoc File::Spec
$infile = ”Intro.pl";
# $! Contains the system error message in case of an open failure
open IN, $infile or die "$infile: $!\n"; #no parens around open, || not o.k.
open (OUT, ">$tmpfile") || die "$tmpfile: $!\n";
while (<IN>) {
# The file is read line by line into $_
  print OUT $_;
# Print a dot every ten lines
  print "." if $. % 10 == 0;
}
# $. Is reset after the close command, therefore use it before
        ###
print "\n$. lines copied to $tmpfile\n";
close IN;
close OUT;
# Remove the copied file
unlink $tmpfile;
```

Visibility of Variables (Scopes)

- By default all variables are global (in package main)

  `$a` is identical to `$main::a`

  therefore also visible in all blocks and subroutines

- Default Package name can be changed (see later)
- Values of global Variables can be hidden

  `local $a#` valid in Block, Subroutine, File, eval

  Variable gets new value within current scope, old value is restored automatically when scope is left

  for magic Variables the only mechanism to limit visibility

Lexical Variables using my

- Definition of lexical Variables with **`my`**
- Lexical Variables only visible in current block, while global and local variables still visible in subroutines
- Variables declared with **`my`** truly local (private)

```
my $a = 1; my ($b, $c); my %hash;
```

- Analogy to auto Variables in C
- Preference of **`my`** over **`local`** is good practice(faster and safer)

The our keyword

- New with perl 5.6
- Perl can restrict the use of global variables

```
use strict;
```

- Prior to perl 5.6 global variables had then either to be fully qualified, e.g. `$main::debug` or be predeclared with

```
use strict vars ($var1, $var2, ...);
```

- With perl 5.6 this construction becomes

```
our ($var1, $var2, ...);
```

Local and lexical variables

```
$a = 1;
{ # a new block
  local $a;
  $a = 2;
} # here $a is reset to 1
{ my $bb;
  $bb = 1;
} # from here on $bb cannot be accessed anymore
# Usage of private copies of arguments in a subroutine
sub private {
  my ($par1, $par2, $par3) = @_;
  print "\$par1 = $par1\n";
  $par1 = 0; # The value in the calling program remains intact
  $_[0] = 1; # The value in the calling program gets overwritten
}
```

Questions and Answers

- Where can I learn more about special characters?

  Answers are in **`perldoc perlop`**, chapter ‘quote and quote-like ...‘

  The special chars \n \t \e \033 \x1b \u and \l are more frequently used
- Which function ist the equivalent of ” ” and ’ ’?

  Answers are in **`perldoc perlop`**, chapter ‘quote and quote-like ...‘
- What is the Value of ”\n” in Windows?

  It is equivalent to the two chars <CR><LF>
- What are the operators that do bitwise arithmetic?

  `& | ^ ~` (and, or, xor, not)

Questions and Answers (2)

- Where can I learn more about magic Variables?

  Look into **`perldoc perlvar`**

  $_ $| $. $? $! $$ $< $0 $& $` $’ @ARGV @INC and %ENV are found in many programs

- What is the purpose of the functions defined and undef?

  Try **`perldoc -f defined`** and **`perldoc -f undef`**

```
if (defined $a) ...      if (defined &sub) ...

undef $a; undef %h; undef &sub; $a = undef;
```