

# T-(538|725)-MALV, Natural Language Processing The programming language Perl

Hrafn Loftsson<sup>1</sup> Hannes Högni Vilhjálmsón<sup>1</sup>

<sup>1</sup>School of Computer Science, Reykjavik University

September 2009

## 1 Perl

- Background
- Format of a program
- Data types and operators
- Control structures
- File handling and command line arguments
- Regular expressions

## 2 When to use Perl?

## 1 Perl

- Background
- Format of a program
- Data types and operators
- Control structures
- File handling and command line arguments
- Regular expressions

## 2 When to use Perl?

## 1 Perl

### ■ Background

- Format of a program
- Data types and operators
- Control structures
- File handling and command line arguments
- Regular expressions

## 2 When to use Perl?

## Background

- PERL = "Practical Extraction and Report Language"
- First version appeared in 1987: Larry Wall
- General purpose tool for Unix-like systems
- More powerful than scripting tools and simpler than C
- Emphasis on fast text processing
- An interpreted language
- Obtaining Perl: <http://www.perl.org/>
  - Part of most Linux installations. Part of Cygwin.

## 1 Perl

- Background
- **Format of a program**
- Data types and operators
- Control structures
- File handling and command line arguments
- Regular expressions

## 2 When to use Perl?

# Hello world

The code: hello.pl

```
use strict;  
use warnings;  
print "What is your name? ";  
my $name = <STDIN>;  
print "Pleased to meet you ", $name;
```

Running it

```
perl hello.pl
```

## The format of a program

- An ordinary text file (\*.pl) with a sequence of Perl statements
- White spaces are ignored (except in strings)
- Each statements ends with a semicolon (";")
- Everything after "#" is ignored (comments)

## A short program

```
#Prints text
print "This is a text\n",
      "which occupies two lines\n";
```



## 1 Perl

- Background
- Format of a program
- **Data types and operators**
- Control structures
- File handling and command line arguments
- Regular expressions

## 2 When to use Perl?

## Scalars

- The most basic data type in Perl
- Can be strings of characters or numbers
- Perl numbers are stored internally as floating-point values
- Scalar variables always start with "\$" (dollar)
- Perl does automatic conversion between numbers and strings
- Perl does “scalar interpolation”
- Scalars are dynamically typed

# Working with scalars

## Example

```
$a=2;  
$b=6;  
$c=$a.$b;  
$d=$c/2;  
print "$d\n"; # Scalar interpolation
```

## Example

- `$name='john'; print "My name is:\t$name\n";`

# Working with scalars

## Undefined variables

```
use strict;  
use warnings;  
  
my $hasValue = "Hello";  
my $hasNoValue;  
  
print "$hasValue $hasNoValue\n";
```

# Numerical operators

```
use strict;  
my $x = 5 * 2 + 3; # $x is 13  
my $y = 2 * $x / 4; # $y is 6.5  
my $z = (2 ** 6) ** 2; # $z is 4096  
my $a = ($z - 96) * 2; # $a is 8000  
my $b = $x % 5; # 3, 13 modulo 5
```

# Comparison operators

Operation	Numeric version	String version
less than	<	lt
less than or equal to	<=	le
greater than	>	gt
greater than or equal to	>=	ge
equal to	==	eq
not equal to	!=	ne

# Other data types

## Arrays

- A *list* is an ordered collection of scalars
- An array is a variable that contains a list
- Array variables start with "@" (at)
- Each element of a list is a separate scalar variable with an independent scalar value

## Example

- `@g=('john',25); print "$g[0] is $g[1] years old\n";`
- `@a=(1,2); @b=(3,4); @c=(@a,@b);`

# Other data types

## Arrays

- Perl arrays grow and shrink dynamically as needed
- You can easily mix different types of scalars within the same array

## A scalar variable associated with arrays

- `$#array`
- This variable contains the subscript of the last element in the array
- `$array[$#array]` # is the last element of the array
- The length of the array is always `$#array + 1`



# Array scalar variable

```
use strict;
my @someStuff = qw /Hello and welcome/;
# @someStuff: an array of 3 elements
print "@someStuff\n";
$#someStuff = 0; # @someStuff now is simply ("Hello")
print "@someStuff\n";
$someStuff[1] = "Joe"; # @someStuff is ("Hello", "Joe")
print "@someStuff\n";
$#someStuff = -1; # @someStuff is now empty
@someStuff = ();
print "@someStuff\n";
```

# Other data types

## Associative Array

- Same as a hash table
- Like an array. Instead of indexing the values by number, values are looked up by strings.
- Strings are *keys*. Hashes start with “%”

## Example

```
%english=('epli' => 'apple', 'pera' => 'pear');  
print $english{'epli'}, "\n";  
print keys(%english), "\n";  
print values(%english), "\n";
```

## 1 Perl

- Background
- Format of a program
- Data types and operators
- **Control structures**
- File handling and command line arguments
- Regular expressions

## 2 When to use Perl?

# Control structures

- if statement
- while/until statement
- for statement

## If statement

```
print "What is your name? ";
my $name = <STDIN>;
chomp($name); # remove the newline character
if ($name gt 'Fred') {
print "$name" comes after Fred\n";
}
else {
print "$name does not come after Fred\n";
print "Maybe it's the same string, in fact\n";
}
```

# Control structures

## foreach statement

```
use strict;  
my @collection = qw/hat shoes shirts shorts /;  
foreach my $item (@collection) {  
    print "$item\n";  
}
```

# Control structures

## foreach statement

```
use strict;  
my %english=('epli' => 'apple', 'pera' => 'pear');  
foreach my $key (keys %english) {  
    print "$english{$key}\n";  
}
```

## 1 Perl

- Background
- Format of a program
- Data types and operators
- Control structures
- **File handling and command line arguments**
- Regular expressions

## 2 When to use Perl?

# File handling

## A file handle

- A string (in upper case) which points to a file
- First associated with a file name and then used for reading and writing
- Perl has already associated `STDIN`, `STDOUT` and `STDERR`

## Example

```
open(INPUT, "test.txt");
open(OUTPUT, ">results.txt");
while (<INPUT>) { # Each line saved in the variable $_
    print OUTPUT $_;
}
close(INPUT); close(OUTPUT);
```





# Command line arguments

```
# Input: A filename and 2 numbers (x,y) as parameters
# Copies lines number x to y
# Output is written to standard output
```

```
$file = shift(@ARGV);           # get the file name
$start = shift(@ARGV);         # first line
$last = shift(@ARGV);         # last line
open(INFILE, "$file");        # open the file
$num = 1;
```

```
while (<INFILE>) {
    if ($num >= $start && $num <= $last) {
        print $_;
    }
    $num++;
}
close(INFILE);
```

## 1 Perl

- Background
- Format of a program
- Data types and operators
- Control structures
- File handling and command line arguments
- Regular expressions

## 2 When to use Perl?

# Regular expressions

## Text processing

- Regular expressions are the core of Perl's text processing construct
- We use them to do pattern matching on text documents
- Three main usages:
  - Matching: **m**/regex/modifiers
  - Substitution: **s**/regex/replacement/modifiers
  - Translation: **tr**/search\_list/replacement\_list/modifiers

# Examples

## Simple match

```
while(<STDIN>) {  
    print if(m/ab*c/);  
}
```

## Simple match – case insensitive

```
while(<STDIN>) {  
    print if(m/ab*c/i);  
}
```

## Substitution and translation

```
while($line = <STDIN>) {  
    if($line =~ m/ab*c/i) {  
        $line =~ s/ab*c/ABC/g;  
    } else {  
        $line =~ tr/a-z/A-Z/;  
    }  
    print $line;  
}
```

# Grouping and parsing

## Referring to a matched string

- One often needs to refer to a matched string, for example, to see if it repeats itself or to print it
- A matched string can be “cached” by using “()” and then referred to by a number within or outside the expression

## Example

```
while ($line = <STDIN>) {  
    while ($line =~ m/\$ *([0-9]+)\.?( [0-9]*)/g) {  
        print "Dollars: ", $1, " Cents: ", $2, "\n";  
    }  
}
```

# The split function

## An extremely useful function

- Splits up a string and places it into an array
- The function uses a regular expression
- See <http://www.comp.leeds.ac.uk/Perl/split.html>

## 1 Perl

- Background
- Format of a program
- Data types and operators
- Control structures
- File handling and command line arguments
- Regular expressions

## 2 When to use Perl?



# When should one use Perl?

## “Quick and dirty”

- When you need to write a “quick and dirty” program in a hurry!
- In most cases, Perl is use to develop a program which takes less than 1 hour to design, write and test.

## When processing text

- Great support for regular expression
- Simple file handling
- Useful data structures.

# Concordance (í. Orðstöðulykill)

- Concordances consist of text excerpts centered on a specific word and surrounded by a limited number of words before and after it.
- The most common concordances are so-called KWIC (Key Word In Context), in which each example of a key word stands in the middle of a line together with context words preceding and following it in the text .
- <http://ysomeya.hp.infoseek.co.jp/>
- <http://www.lexis.hi.is/corpus/leit.pl>

# Implementing concordances in Perl– concordance.pl

```
use warnings;
($file_name, $pattern, $width) = @ARGV;
open(FILE, "$file_name") || die "Could not open $file_name."
while ($line = <FILE>) {
    $text .= $line;
}

# new lines are replaced by spaces
$text =~ s/\n/ /g;

# matches the pattern with 0..width to the right and left
while ($text =~ m/(.{0,$width}$pattern.{0,$width})/g) {
    print "$1\n";    # $1 contains the match
}
}
```

# Implementing concordances in Perl– concordance.pl

- `perl concordance.pl visindavefur2.txt vegna 15 > concordance.out`
- `perl concordance.pl visindavefur2.txt einnig 20 > concordance.out`