

COMP284 Scripting Languages

Lecture 2: Perl (Part 1) Handouts (8 on 1)

Ullrich Hustadt

Department of Computer Science
School of Electrical Engineering, Electronics, and Computer Science
University of Liverpool

Perl: Overview

Applications

Perl: Applications

- Applications written in Perl
 - **Movable Type** – web publishing platform
<http://www.movabletype.org/>
 - **Request Tracker** – issue tracking system
<http://bestpractical.com/rt/>
 - **Slash** – database-driven web application server
<http://sourceforge.net/projects/slashcode/>

COMP284 Scripting Languages

Lecture 2

Slide L2 – 4

Contents

- 1 Perl: Overview
 - History
 - Applications
 - Java vs Perl
- 2 Scalars
 - Definition
 - Integers and Floating-point numbers
 - Strings
 - 'Booleans'
 - Comparisons
- 3 Variables, Constants, and Assignments
 - Variables
 - Constants
 - Assignments
 - Variable interpolation

COMP284 Scripting Languages

Lecture 2

Slide L2 – 1

Perl: Overview

Applications

Perl: Applications

- Organisations using Perl
 - **Amazon** – online retailer
<http://www.amazon.co.uk>
 - **BBC** – TV/Radio/Online entertainment and journalism
<http://www.bbc.co.uk>
 - **Booking.com** – hotel bookings
<http://www.booking.com>
 - **craigslist** – classified ads
<http://www.craigslist.org>
 - **IMDb** – movie database
<http://www.imdb.com>
 - **Monsanto** – agriculture/food/culture
<http://www.monsanto.co.uk/>
 - **Slashdot** – technology related news

COMP284 Scripting Languages

Lecture 2

Slide L2 – 4

Perl

- Originally developed by **Larry Wall** in 1987
Perl 6 was released in December 2015
- Borrows features from
 - **C**
imperative language with variables, expressions, assignment statements, blocks of statements, control structures, and procedures / functions
 - **Lisp**
lists, list operations, functions as first-class citizens
 - **AWK** (pattern scanning and processing language)
hashes / associative arrays, regular expressions
 - **sed** (stream editor for filtering and transforming text)
regular expressions and substitution s///
 - **Shell**
use of **sigils** to indicate **type** (\$ – scalar, @ – array, % – hash, & – procedure)
 - **Object-oriented programming languages**
classes/packages, inheritance, methods

COMP284 Scripting Languages

Lecture 2

Slide L2 – 2

Java versus Perl: Java

```
1 // HelloWorld.java
2
3
4 */
5 public class HelloWorld {
6     // -----METHODS-----
7     /* Main Method */
8     public static void main(String[] args) {
9         System.out.println("Hello World");
10    }
11 }
```

Edit-compile-run cycle:

- 1 Edit and save as HelloWorld.java
- 2 Compile using javac HelloWorld.java
- 3 Run using java HelloWorld

COMP284 Scripting Languages

Lecture 2

Slide L2 – 6

Perl: Uses and applications

- Main application areas of Perl
 - **text processing**
~ easier and more powerful than sed or awk
 - **system administration**
~ easier and more powerful than shell scripts
- Other application areas
 - web programming
 - code generation
 - bioinformatics
 - linguistics
 - testing and quality assurance

COMP284 Scripting Languages

Lecture 2

Slide L2 – 3

Perl: Overview

Java vs Perl

Java versus Perl: Perl

```
1 #!/usr/bin/perl
2 # Author: Ullrich Hustadt
3 # The HelloWorld script implements an application
4 # that prints out "Hello World".
5
6 print "Hello World\n";
```

Edit-run cycle:

- 1 Edit and save as HelloWorld
- 2 Run using perl HelloWorld

-
- 1 Edit and save as HelloWorld
 - 2 Make it executable
chmod u+x HelloWorld
This only needs to be done once!
 - 3 Run using ./HelloWorld

COMP284 Scripting Languages

Lecture 2

Slide L2 – 7

<div>Perl: OverviewJava vs Perl</div> <div>Perl</div> <ul style="list-style-type: none"> Perl borrows features from a wide range of programming languages including imperative, object-oriented and functional languages Advantage: Programmers have a choice of programming styles Disadvantage: Programmers have a choice of programming styles Perl makes it easy to write completely incomprehensible code ~ Documenting and commenting Perl code is very important <div>COMP284 Scripting LanguagesLecture 2Slide L2 – 8</div>	<div>Perl: OverviewJava vs Perl</div> <div>Perl scripts</div> <ul style="list-style-type: none"> Perl statements include <ul style="list-style-type: none"> Assignments Control structures Every statement returns a value Perl data types include <ul style="list-style-type: none"> Scalars Arrays / Lists Hashes / Associative arrays Perl expressions are constructed from values and variables using operators and subroutines <ul style="list-style-type: none"> Perl expressions can have side-effects (evaluation of an expression can change the program state) Every expression can be turned into a statement by adding a semi-colon <div>COMP284 Scripting LanguagesLecture 2Slide L2 – 12</div>
<div>Perl: OverviewJava vs Perl</div> <div>Perl</div> <ul style="list-style-type: none"> Perl makes it easy to write completely incomprehensible code ~ Documenting and commenting Perl code is very important <pre> 1 #!/usr/bin/perl 2 # Authors: Schwartz et al. / Ullrich Hustadt 3 # Text manipulation using regular expressions 4 # 5 # Retrieve the Perl documentation of function 'atan2' 6 @lines = `perl-doc -u -f atan2`; 7 8 # Go through the lines of the documentation, turn all text 9 # between angled brackets to uppercase and remove the 10 # character in front of the opening angled bracket, then 11 # print the result 12 foreach (@lines) { 13 s/\w<([^\>]+)>/\U\$1/g; 14 print; 15 }</pre> <p>In the example, there are more lines of comments than t</p> <div>COMP284 Scripting LanguagesLecture 2Slide L2 – 9</div>	<div>ScalarsDefinition</div> <div>Scalar data</div> <ul style="list-style-type: none"> A scalar is the simplest type of data in Perl A scalar is either <ul style="list-style-type: none"> an integer number 0 2012 -40 1_263_978 a floating-point number 1.25 256.0 -12e19 2.4e-10 a string 'hello world' "hello world\n" Note: <ul style="list-style-type: none"> There is no integer type, string type, etc. There are no boolean constants (true / false) <div>COMP284 Scripting LanguagesLecture 2Slide L2 – 10</div>
<div>Perl: OverviewJava vs Perl</div> <div>Perl for Java programmers</div> <ul style="list-style-type: none"> In the following we will consider various constructs of the Perl programming language <ul style="list-style-type: none"> numbers, strings variables, constants assignments control structures These will often be explained with reference to Java ('like Java', 'unlike Java') Note that Perl predates Java ~ common constructs are almost always inherited by both languages from the programming language C <div>COMP284 Scripting LanguagesLecture 2Slide L2 – 10</div>	<div>Integers and Floating-point numbers</div> <ul style="list-style-type: none"> Mathematical functions <ul style="list-style-type: none"> random(<i>number</i>) random number between 0 and <i>number</i> sqrt(<i>number</i>) square root Additional functions are available via the POSIX module <ul style="list-style-type: none"> ceil(<i>number</i>) round fractions up floor(<i>number</i>) round fractions down Note: There is no pre-defined round function <pre> use POSIX; print ceil(4.3); // prints '5' print floor(4.3); // prints '4' </pre> Remember: Floating-point arithmetic has its peculiarities David Goldberg: What Every Computer Scientist Should Know About Floating-Point Arithmetic. Computing Surveys 23(1):5–48. http://perso.ens-lyon.fr/jean-michel.muller/goldberg.pdf <div>COMP284 Scripting LanguagesLecture 2Slide L2 – 14</div>
<div>Perl: OverviewJava vs Perl</div> <div>Perl scripts</div> <ul style="list-style-type: none"> A Perl script consists of one or more statements and comments ~ there is no need for a main function (or classes) <ul style="list-style-type: none"> Statements end in a semi-colon Whitespace before and in between statements is irrelevant (This does not mean its irrelevant to someone reading your code) Comments start with a hash symbol # and run to the end of the line Comments should precede the code they are referring to <div>COMP284 Scripting LanguagesLecture 2Slide L2 – 11</div>	<div>ScalarsIntegers and Floating-point numbers</div> <div>Mathematical functions and Error handling</div> <ul style="list-style-type: none"> Perl, PHP and JavaScript differ in the way they deal with applications of mathematical functions that do not produce a number In Perl we have <ul style="list-style-type: none"> log(0) produces an error message: Can't take log of 0 sqrt(-1) produces an error message: Can't take sqrt of -1 1/0 produces an error message: Illegal division by zero 0/0 produces an error message: Illegal division by zero and execution of a script terminates when an error occurs A possible way to perform error handling in Perl is as follows: <pre> eval { ...run the code here... } # try 1; } or do { ...handle the error here using \$@... } # catch }; </pre> The special variable \$@ contains the Perl syntax or routine error message from the last eval, do-FILE, or require command <div>COMP284 Scripting LanguagesLecture 2Slide L2 – 15</div>

Scalars

Strings

Strings

Perl distinguishes between

- single-quoted strings and
- double-quoted strings

single-quoted strings (‘taken literally’)	double-quoted strings (‘interpreted’/‘evaluated’)
<code>'hello'</code> ~ hello	<code>"hello"</code> ~ hello
<code>'don't'</code> ~ don't	<code>"don't"</code> ~ don't
<code>'"hello"'</code> ~ "hello"	<code>"\"hello\""</code> ~ "hello"
<code>'backslash\\'</code> ~ backslash\	<code>"backslash\\"</code> ~ backslash\
<code>'glass\\table'</code> ~ glass\table	<code>"glass\\table"</code> ~ glass\table
<code>'glass\table'</code> ~ glass\table	<code>"glass\table"</code> ~ glass able

In Java, **single quotes** are used for single characters and **double quotes** for strings

COMP284 Scripting LanguagesLecture 2Slide L2 – 16

Scalars

Strings

String operators and automatic conversion

- Two basic operations on strings are
 - string concatenation**
`"hello" . "world"` ~ "helloworld"
`"hello" . 'world'` ~ 'hello_world'
`"\Uhello" . ' \LWORLD '` ~ 'HELLO\LWORLD'
 - string repetition** x:
`"hello_" x 3` ~ "hello_hello_hello_"
- These operations can be combined
`"hello_" . "world_" x 2` ~ "hello_world_world_"
- Perl automatically converts between strings and numbers**
`2 . "_worlds"` ~ "2_worlds"
`"2" * 3` ~ 6
`2e-1 x 3` ~ "0.20.20.2" ("0.2" repeated three times)
`"hello" * 3` ~ 0

COMP284 Scripting LanguagesLecture 2Slide L2 – 20

Scalars

Strings

Double-quoted string backslash escapes

- In a single-quoted string `\t` is simply a string consisting of `\` and `t`
- In a double-quoted string `\t` and other **backslash escapes** have the following meaning

Construct	Meaning
<code>\n</code>	Logical Newline (actual character is platform dependent)
<code>\f</code>	Formfeed
<code>\r</code>	Return
<code>\t</code>	Tab
<code>\l</code>	Lower case next letter
<code>\L</code>	Lower case all following letters until <code>\E</code>
<code>\u</code>	Upper case next letter
<code>\U</code>	Upper case all following letters until <code>\E</code>
<code>\Q</code>	Quote non-word characters by adding a backslash until <code>\E</code>
<code>\E</code>	End <code>\L</code> , <code>\U</code> , <code>\Q</code>

COMP284 Scripting LanguagesLecture 2Slide L2 – 17

Scalars

'Booleans'

'Booleans'

- Unlike Java, Perl does **not** have a **boolean datatype**
- Instead the values
 - `0` # zero
 - `''` # empty string
 - `'0'` # string consisting of zero
 - `undef` # undefined
 - `()` # empty list

all represent **false** while all other values represent **true**

COMP284 Scripting LanguagesLecture 2Slide L2 – 18

Scalars

Strings

UTF-8

- Perl supports **UTF-8** character encodings which give you access to non-ASCII characters
- The pragma
`use utf8;`
allows you to use UTF-8 encoded characters in Perl scripts
- The function call
`binmode(STDIN, ":encoding(UTF-8)");`
`binmode(STDOUT, ":encoding(UTF-8)");`
ensures that UTF-8 characters are read correctly from STDIN and printed correctly to STDOUT
- The `Unicode::Normalize` module enables correct **decomposition** of strings containing UTF-8 encoded characters
`use Unicode::Normalize;`

COMP284 Scripting LanguagesLecture 2Slide L2 – 18

Scalars

'Boolean operators'

'Boolean operators'

Perl operators as Java: `&&`, `||`, `!`

A	B	(A && B)	A	B	(A B)
true	true	B (true)	true	true	A (true)
true	false	B (false)	true	false	A (true)
false	true	A (false)	false	true	B (true)
false	false	A (false)	false	false	B (false)

A	(! A)
true	'' (false)
false	1 (true)

- Note that this means that `&&` and `||` are **not commutative**, that is, `(A && B)` is not the same as `(B && A)`
`($denom != 0) && ($num / $denom > 10)`

COMP284 Scripting LanguagesLecture 2Slide L2 – 22

Scalars

Strings

UTF-8

Example:

```
binmode(STDOUT, ":utf8");
print "\x{4f60}\x{597d}\x{4e16}\x{754c}\n"; # chinese
print "\x{062d}\x{f6f0}\n"; # arabic
```

For further details see Schwartz et al., Appendix C

COMP284 Scripting LanguagesLecture 2Slide L2 – 19

Scalars

Comparisons

Comparison operators

Perl distinguishes between **numeric comparison** and **string comparison**

Comparison	Numeric	String
Equal	<code>==</code>	<code>eq</code>
Not equal	<code>!=</code>	<code>ne</code>
Less than	<code><</code>	<code>lt</code>
Greater than	<code>></code>	<code>gt</code>
Less than or equal to	<code><=</code>	<code>le</code>
Greater than or equal to	<code>>=</code>	<code>ge</code>

Examples

```
35 == 35.0 # true
'35' eq '35.0' # false
'35' == '35.0' # true
35 < 35.0 # false
'35' lt '35.0' # true
'ABC' eq "\Uabc" # true
```

COMP284 Scripting LanguagesLecture 2Slide L2 – 23

Variables, Constants, and Assignments

Variables

Scalar variables

- Scalar variables start with \$ followed by a Perl identifier
- A Perl identifier consists of letters, digits, and underscores, but cannot start with a digit
Perl identifiers are case sensitive
- In Perl, a variable does not have to be declared before it can be used
- Scalar variables can store any scalar value (there are no 'integer variables' versus 'string variables')

COMP284 Scripting LanguagesLecture 2Slide L2 – 24

Variables, Constants, and Assignments

Assignments

Assignments

- Just like Java, Perl uses the equality sign = for assignments:

```
$student_id = 200846369;  
$name      = "Jan_Olsen";  
$student_id = "E00481370";
```

But no type declaration is required and the same variable can hold a number at one point and a string at another
- An assignment also returns a value, namely (the final value of) the variable on the left
~ enables us to use an assignment as an expressions

Example:

```
$b = ($a = 0) + 1;  
# $a has value 0  
# $b has value 1
```

COMP284 Scripting LanguagesLecture 2Slide L2 – 28

Variables, Constants, and Assignments

Variables

Scalar variables

- A variable also does not have to be initialised before it can be used, although initialisation is a good idea
- Uninitialised variables have the special value undef
However, undef acts like 0 for numeric variables and like '' for string variables if an uninitialised variable is used in an arithmetic or string operation
- To test whether a variable has value undef use the routine defined

```
$s1 = "";  
print '$s1 eq undef: ', ($s1 eq undef) ? 'TRUE' : 'FALSE', "\n";  
print '$s1 defined: ', (defined $s1) ? 'TRUE' : 'FALSE', "\n";  
print '$s2 defined: ', (defined $s2) ? 'TRUE' : 'FALSE', "\n";  
  
$s1 eq undef: TRUE  
$s1 defined: TRUE  
$s2 defined: FALSE
```

COMP284 Scripting LanguagesLecture 2Slide L2 – 25

Variables, Constants, and Assignments

Assignments

Binary assignments

There are also binary assignment operators that serve as shortcuts for arithmetic and string operations

Binary assignment	Equivalent assignment
\$a += \$b	\$a = \$a + \$b
\$a -= \$b	\$a = \$a - \$b
\$a *= \$b	\$a = \$a * \$b
\$a /= \$b	\$a = \$a / \$b
\$a %= \$b	\$a = \$a % \$b
\$a **= \$b	\$a = \$a ** \$b
\$a .= \$b	\$a = \$a . \$b

Example:

```
# Convert Fahrenheit to Celsius:  
# Subtract 32, then multiply by 5, then divide by 9  
# temperature in Fahrenheit  
# converted to Celsius
```

COMP284 Scripting LanguagesLecture 2Slide L2 – 25

Variables, Constants, and Assignments

Variables

Special Variables

- Perl has a lot of 'pre-defined' variables that have a particular meaning and serve a particular purpose

Variable	Explanation
\$_	The default or implicit variable
@_	Subroutine parameters
\$a, \$b	sort comparison routine variables
\$_	the string matched by the last successful pattern match
\$/	input record separator, newline by default
\$\	output record separator, undef by default
\$]	version of Perl used

- For a full list see <https://perldoc.perl.org/perlvar.html#SPECIAL-VARIABLES>

COMP284 Scripting LanguagesLecture 2Slide L2 – 26

Variables, Constants, and Assignments

Assignments

Variable declarations

- ```
use strict;
```

enforces that all variables must be declared before their use, otherwise a compile time error is raised

Example:

```
use strict;
$studentsOnCOMP284 = 133;

Global symbol "$studentOnCOMP284" requires explicit
package name at ./script line 2.
Execution of ./script aborted due to compilation errors.

use strict;
my $studentsOnCOMP281;
$studentsOnCOMP281 = 154;
my $studentsOnCOMP283 = 53;
```

COMP284 Scripting LanguagesLecture 2Slide L2 – 30

Variables, Constants, and Assignments

Constants

Constants

Perl offers three different ways to declare constants

- Using the constant pragma:

```
use constant PI => 3.14159265359;
```

(A pragma is a module which influences some aspect of the compile time or run time behaviour of Perl)
- Using the Readonly module:

```
use Readonly;
Readonly $PI => 3.14159265359;
```
- Using the Const::Fast module:

```
use Const::Fast;
const $PI => 3.14159265359;
```

With our current Perl installation only constant works  
~ variable interpolation with constants does not work

COMP284 Scripting LanguagesLecture 2Slide L2 – 27

Variables, Constants, and Assignments

Variable interpolation

Variable interpolation

Any scalar variable name in a double quoted string is (automatically) replaced by its current value

Example:

```
$actor = "Jeff_Bridges";
$prize = "Academy_Award_for_Best_Actor";
$year = 2010;
print "1: ", $actor, " won the ", $prize, " in ", $year, "\n";
print "2: $actor won the $prize in $year\n";
```

Output:

```
1: Jeff Bridges won the Academy Award for Best Actor in 2010
2: Jeff Bridges won the Academy Award for Best Actor in 2010
```

COMP284 Scripting LanguagesLecture 2Slide L2 – 31

## Revision

Read

- Chapter 2: Scalar Data

of

R. L. Schwartz, brian d foy, T. Phoenix:

Learning Perl.

O'Reilly, 2011.

Harold Cohen Library: 518.579.86.S39 or e-book

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro