# COMP284 Scripting Languages

Lecture 2: Perl (Part 1)

Handouts

Department of Computer
School of Electrical Engineering, Electronics, an
University of Liverpo

## Contents

## 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, bloc
  - Lisp
    lists l
  - AWK (pattern scanning and processing language
    hashes / associative arrays, regular expressions
  - sed (stream editor for filtering and transforming tex
    regular expressions and substitution s///
  - Shell
    use of sigils to indicate type ($ – scalar, @ – array, % – hash, & – procedure)
  - Object-oriented programming languages
    classes/packages, inheritance, methods

## 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 a
  - web
  - cod
  - bioinformatics
  - linguistics
  - testing and quality assurance

## Perl: Applications

- Applications written in Perl
  - Movable Type – web publishing platform
    `http://www.movabletype.org/`
  - Request Tracker – issue tracking system
    `http://bestpractical.com/rt/`
  - Slas – database-d
    `htt`

## 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`
  - Boo
    `htt`

  - crai – clas
    `http://www.craigslist.org`
  - IMDb – movie database
    `http://www.imdb.com`

  - Monsanto – agriculture/biotech
    `http://www.monsanto.co.uk/`
  - Slashdot – technology related news
    `http://slashdot.org`

## Java versus Perl: Java

```
/* Author: Clare Dixon
 * The HelloWorld class implements an application
 * that prints out "Hello World".
 */
public class HelloWorld {
  // ---
  /* Mai   M
  pub
    S
  }
}
```

Edit-compile-run cycle:

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

## Java versus Perl: Perl

```
#!/usr/bin/perl
# Author: Ullrich Hustadt
# The HelloWorld script implements an application
# that prints out "Hello World".

print "He
```

Edit-run c

1. Edit and save as `He`
2. Run using `pe`

---

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`

## Perl

- 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
- Disad
- Perl m
  ⤳ Documenting and commenting Perl code is very important

## Perl

- Perl makes it easy to write completely incomprehensible code
  ⇝ Documenting and commenting Perl code is very important

```
#!/usr/bin/perl
# Authors: Schwartz et al. / Ullrich Hustadt
# Text manip
#
# Retrieve t
@lines = `pe

# Go through the lines of the documentation, turn all text
# between angled brackets to uppercase and remove the
# character in front of the opening angled bracket, then
# print the result
foreach (@lines) {
    s/\w<([^\>]+)>/\U$1/g;
    print;
}
```

In the example, there are more lines of comments than there are lines of code

## Perl for Java programmers

- In the following we will consider various constructs of the Perl programming language
  - numbers, strings
  - variables, constants
  - assi
  - con
- 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

## Perl scripts

- A Perl script consists of one or more statements and comments
  ⇝ there is no need for a main function (or classes)
- Statements end in a semi-colon
- Whitespace before and in between statements is irrelevant
  (Th
- Co
- Co

## Perl scripts

- Perl statements include
  - Assignments
  - Control structures

  Every statement returns a value

- Perl da
  - Scal
  - Arr
  - Hashes / Associative arrays

- Perl expressions are constructed from values and v
  operators and subroutines

  - 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

## Scalar data

- A scalar is the simplest type of data in Perl
- A scalar is either
  - an integer number
    `0   2012   -40   1_263_978`
  - a floa
    `1.2   2`
  - a str
    `'hello world'   "hello world\n"`
- Note:
  - There is no ‘integer type’, ‘string type’ etc
  - There are no boolean constants (true / false)

## Integers and Floating-point numbers

- Perl provides a wide range of pre-defined mathematical functions

| | |
|---|---|
| abs(*number*) | absolute value |
| log(*number*) | natural logarithm |
| random(*number*) | random number between 0 and *number* |
| sqrt(*number*) | square root |

- Additi

ceil

floo

Note: There is no pre-defined round fu

```
use POSIX;
print ceil(4.3); // prints '5'
print floor(4.3); // prints '4'
```

- 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`

## Mathematical functions and Error handling

- 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

  - `log(0)` produces an error message: `Can't take log of 0`
  - `sqr`
  - `1/0`
  - `0/0`

  and execution of a script terminates when an error oc

- A possible way to perform error handling in Perl is as fol

```
eval {      ...run the code here...
            1;
} or do { ...handle the error here using $@... # catch
};
```

The special variable `$@` contains the Perl syntax or routine error message from the last `eval`, `do`-FILE, or `require` command

## Strings

Perl distinguishes between

- single-quoted strings and
- double-quoted strings

| single-q ('taken lit | | | ') |
|---|---|---|---|
| 'hello | | | llo |
| 'don\' | | | n't |
| '"hello"' | ⇝ "hello" | "\"" | " |
| 'backslash\\' | ⇝ backslash\ | "ba | ash\ |
| 'glass\\table' | ⇝ glass\table | "gl | table |
| 'glass\table' | ⇝ glass\table | "glass\table" | ⇝ glass able |

In Java, single quotes are used for single characters and double quotes for 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 |
|---|---|
| `\n` | [illegible] |
| `\f` | [illegible] |
| `\r` | [illegible] |
| `\t` | Tab |
| `\l` | Lower case next letter |
| `\L` | Lower case all following letters unt[illegible] |
| `\u` | Upper case next letter |
| `\U` | Upper case all following letters until `\E` |
| `\Q` | Quote non-word characters by adding a backslash until `\E` |
| `\E` | End `\L`, `\U`, `\Q` |

## UTF-8

- Perl supports UTF-8 character encodings which give you access to non-ASCII characters
- The pragma

```
use utf8;
```

  allows

- The fu

```
binmode(STDIN,  ":encoding(UTF-8)");
binmode(STDOUT, ":encoding(UTF-8)");
```

  ensures that UTF-8 characters are read correctly f
  printed correctly to STDOUT

- The `Unicode::Normalize` module enables correct decomposition of strings containing UTF-8 encoded characters

```
use Unicode::Normalize;
```

## UTF-8

Example:

```
binmode(STDOUT, ":utf8");
print "\x{4f60}\x{597d}\x{4e16}\x{754c}\n";     # chinese
print "\x{062d}\x{fef0}\n";                      # arabic
```

For furth

## String operators and automatic conversion

- Two basic operations on strings are
  - string concatenation .

```
"hello" . "world"                  ⇝ "helloworld"
"hello" . '␣' . "world"            ⇝ 'hello␣world'
"\Uhello" . '␣\LWORLD'               'HELLO␣\LWORLD'
```

  - stri

```
"he
```

- These operations can be combined

```
"hello␣" . "world␣" x 2  ⇝                    ␣"
```

- Perl automatically converts between strings an

```
2 . "␣worlds"  ⇝  "2␣worlds"
"2" * 3        ⇝  6
2e-1 x 3       ⇝  "0.20.20.2" ("0.2" repeated three times)
"hello"* 3     ⇝  0
```

## 'Booleans'

- Unlike Java, Perl does not have a boolean datatype
- Instead the values

```
0     # zero
''    # empty string
'0'
undef
()
```

all represent *false* while all other values repres

## 'Boolean operators'

- Perl offers the same short-circuit boolean operators as Java: `&&`, `||`, `!`
  Alternatively, `and`, `or`, `not` can be used

| `A` | `B` | `(A && B)` |
|---|---|---|
| *true* | *true* | `B` (*true*) |
| *true* | | |
| *false* | | |
| *false* | | |

| `A` | `B` | `(A \|\| B)` |
|---|---|---|
| *true* | *true* | `A` (*true*) |
| | | |
| | | |
| | | |

| `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)
```

## Comparison operators

Perl distinguishes between numeric comparison and string comparison

| Comparison | Numeric | String |
|---|---|---|
| Equal | == | eq |
| Not equal | != | ne |
| Greater than or equal to | | ge |

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
```

## 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
- Scalar
  (there

## 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

  like '

  if an uni
- To test whether a variable has value un

```
$s1 = "";
print '$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
```

## Special Variables

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

| Variable | Explanation |
|---|---|
| `$_` | The default or implicit variable |
| `@_` | |
| `$a, $` | |
| `$&` | |
| `$/` | input record separator, newline by default |
| `$\` | output record separator, `un` |
| `$]` | version of Perl used |

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

## 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 compi

- Using t

```
use Read
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

## Assignments

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

```
$student_id = 200846369;
$name       = 'Jan␣Olsen';
$student_id = "E00481370";
```

But no t

numb

- An assi
  namely (the final value of) the variable on the left
  ⇝ enables us to use an assignment as an expression

Example:

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

## 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 + $b` |
| [illegible] | [illegible] |
| [illegible] | [illegible] |
| [illegible] | [illegible] |
| `$a  %= $b` | `$a = $a % $b` |
| `$a **= $b` | `$a = $a ** $` |
| `$a  .= $b` | `$a = $a . $b` |

Example:

```
# Convert Fahrenheit to Celsius:
# Subtract 32, then multiply by 5, then divide by 9
$temperature = 105;               # temperature in Fahrenheit
($temperature -= 32) *= 5/9;      # converted to Celsius
```

## Variable declarations

- In Perl, variables can be declared using the `my` function
  (Remember: This is not a requirement)
- The pragma

```
use strict;
```

enforc

other

Exam

```
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;
```

## Variable interpolation

### Variable interpolation

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

Example

```
$actor = "Jeff [illegible]
$prize = "Acad[illegible]
$year  = 2010;
print "1:␣",$actor,"␣won␣the␣",$prize,"␣in␣",$year,"\
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
```

## Revision

Read

- Cha

of

R. L. Sch

Learning Perl.

O'Reilly, 2011.

Harold Cohen Library: 518.579.86.S39 or e-book