COMS 3101
Programming Languages: Perl

Lecture 1

Fall 2013
Instructor: Ilia Vovsha

http://www.cs.columbia.edu/~vovsha/coms3101/perl

What is Perl?

- Perl is a high-level language initially developed as a scripting language to make report processing easier
- Scripting language designed for “gluing together” computations
- Written in C
- Many features from shell programming, Lisp (lists), AWK (hashes), sed (regular expressions)

Why Perl?

- Powerful built-in text processing functionality
- Easy to extend using modules/packages
- Can embed perl in other languages
- Intuitive, easy to use
- Automatic data-typing, memory management
- Comprehensive Perl Archive Network (CPAN) modules
- Open source development

Course Info – Instructor

- Ilia (Eli) Vovsha
  - Email: iv2121@columbia.edu
  - Office Hours:
    - Friday (1:00 – 2:00pm), TA room (Mudd 122A)
    - Monday (3:00 – 4:00pm), notify by email
- PhD student in Machine Learning (final year)
  - ML is a field in which we develop algorithms/systems with a learning component
  - Major ML applications in Natural Language Processing (NLP)
  - Algorithms + Text Data + Experiments ➔ Perl

Course Info – Goals

- Learn PERL to:
  - Perform simple manipulations on data
  - Handle language and text processing
  - Integrate multiple tools/interfaces
  - Implement prototypes to solve toy problems
- We will focus on the formulation and implementation of (simple) natural language (text) problems

Course Info – Syllabus

- Basics:
  - Variables, data structures, operators
  - Arrays, hashes, lists
- Control Flow:
  - If, while, for
  - Unless, until, foreach
- Subroutines
- Input/Output
- Regular expressions, pattern matching:
  - Matching, substitution
  - Quantifiers, grouping, classes

Course Info – Syllabus

- Array / Hash manipulation:
  - Built-in functions
  - References
  - Array of hashes, hash of hashes
- Packages & Modules:
  - Definition and use of external code
- Object Oriented Programming (OOP):
  - Inheritance, methods, pragmas

Course Info – Grading

- 4 Homeworks (4 x 15%)
    - Posted on Monday morning. Due the following Mon, by start of class
    - See course website for submission instructions
    - Extra HW to make up / replace lowest grade
- Final Project (40%)
    - Solve a toy problem of your choice. Submit a write-up and your code

#!

- Interpreter directive
  - Run interpreter program, pass path as argument
  - Line ignored by interpreter since ‘#’ character is a comment marker
- First line of a any perl script (usually):
  - **#!/usr/local/bin/perl**
- Perl file extension: .pl
- Check syntax without executing script:
  - **$ perl –c test.pl**

Basic Elements

- Variables:
  - Scalars
  - Arrays
  - Hashes
- Operators:
  - Numeric, String, Logical

Scalars

- Singular value denoted by $ (dollar sign)
- No need to declare the exact type of scalar, as perl converts types based on the context
- Common types: integers, strings, floating-point numbers, references
- Examples:
  1. $age = 42;
  2. $pi = 3.14;
  3. $person = “Rob”;
  4. $days[0] = “Sunday”;
  5. $varref = \$person;

Arrays

- Can hold a set of ordered scalars
- Denoted by @
- Examples:

```
1.  @names = ();                     # empty
2.  @names = (“Ben”, “Jen”, “Ken”);  # initialization
3.  @mixed = (123,“Rob”,3.14);       # different types
4.  print  $names[1];                # prints Jen
```

Array Operations: Initializing

- Examples:
  1. @names = (“Ben”, “Jen”, “Ken”); # Strings
  2. @names = (Ben,Jen,Ken); # without quotes
  3. @ages = (37, 34, 4); # numbers
  4. @words = qw(a short sentence); # quoted words
  5. @digits = (0..9); # range
  6. @chars = (a..z); # range
  7. @chars = (ax..bb); # range (ax,ay,az,ba,bb)

Array Operations: Indexing & Slicing

- Examples:

  1. print $names[1], ": ", $ages[1];   # Print '**Jen: 34**'
  2. print $names[-1];   # Print '**Ken**'
  3. @parents = @names[0,1];   # Slice (Ben,Jen)
  4. @schars = @chars[1..3,7,9];   # Slice (b,c,d,h,j)
  5. print $#names;   # Print last index
  6. print @names;   # Print all elements

Hashes

- Can hold a set of unordered scalars indexed by strings (keys)
- Denoted by %
- Examples:

```
1.  %names = ();                                    # empty
2.  %names = (“Ben”, 37, “Jen”, 34, “Ken”, 4);      # initialization
3.  %names = (Ben => 37, Jen => 34, Ken => 4);      # initialization
4.  print  $names{“Ken”};                           # prints 4
```

Hash Operations

- Examples:

```
1. %agehash = (Ben => 37, Jen => 34, Ken => 4);        # initialization
2. $age = $agehash{Ken};                                # $age = 4
3. @p_age = @agehash{Ben,Jen};                          # Slice (37,34)
4. @names = keys %agehash;                   # Random order of keys
5. @vals = values %agehash;                  # Random order of values
```

Operators (Numeric, String)

- Numeric: the usual arithmetic operators
  - {+, -, *, /, %, **}
- String: concatenation, repetition Numeric:
  - {., x}
- Examples:
  1. $n = $a   OP $b;
  2. $n = $a ** $b;
  3. $s = $a . $b;
  4. $s = $a x $b;

Operators (Example)

- Examples:

```
1.  $a = 30;  $b = 2;
2.  print  $a  +  $b;          # 32
3.  print  $a  .  $b;          # 302
4.  print  $a  *  $b;          # 60
5.  print  $a  x  $b;          # 3030
6.  print  “-”  x  $width;          # -------------
```

More Operators

- Assignment:
  1. $s = $a + $b;
  2. $s = $a = $b = 0; # all get 0
  3. $a += 2;
- Unary:
  1. $a++; ++$a;
  2. $a--; --$a;
- Logical:
  1. $s = $a && $b;
  2. $s = $a || $b;
  3. $s = !$a;
  4. $s = $a and $b;
  5. $s = $a or $b; $s = $a xor $b;
  6. $s = not $a;

Concepts

- Quotes
- Interpolation
- Context
- Arrays vs. Lists

Quotes

- Single quotes: use it literally
- Double quotes: interpolate before processing
- Back-quotes (backtick, qx): execute commands
- Examples:
  1. print “$person is \”$age\” years old”;
  2. print ‘Is it worth $50?’;
  3. print `ls –l`;
  4. qx (ls -l)

Interpolation

- Disambiguation among alpha-numeric characters
- Array interpolation
- Escape characters
- Examples:
  1. print “${user}id”;
  2. print @names;   print “@names”;
  3. print “$person is \”$age\” years \t old”;

Context

- Scalar / List
  1. $x = somefun();
  2. @x = somefun();
  3. $x[1] = somefun();   @x[1] = somefun();
- Void: no return value (perl warning)
- Boolean
  - Condition / Expression is evaluated to “true” or “false”
  - False: null string (“”), zero (0), string zero (“0”), undefined value (undef)

Context (Examples)

- Examples:
  1. @family1 = @family2; # list context
  2. ($name1, $name2) = @family; # Slice of array
  3. ($name) = @family; # list, 1st element of array
  4. $num = @family; # scalar, size of array
  5. print @names; # list context (no spaces)
  6. while (@files) { # boolean, checks if array is empty

Arrays vs Lists

- List: ordered set of scalars, separated by commas
- Array: can hold a list
- Examples:
  1. @numbers = (1,2,4,5,8);
  2. print $n1, $n2, $n3;
  3. ($num) = @number; # 1st element of array
  4. $num = (1,2,4,5,8); # last element of list
  5. $num = @number + 1; # scalar, size of array + 1

Control Flow

- Statements:
  - if & unless
  - while & until
  - for & foreach
  - last & next

Statements: if & unless

- ‘if’ Examples:

```
1.  $res  =  $a  < = >  $b;             # returns 0 (==), 1 (>), -1 (<)
2.  If  ($res  ==  0)  {                 # required braces mark blocks
        print  “$a  eq  $b”;  }          #  end of block
    elsif ($res  <  0) {
        print  “$a  le  $b”;  }          #  end of block
    else {
        print  “$a  gt  $b”;  }          #  end of block
```

- ‘unless’ Example:

```
1.  unless ($time  eq  $money)  {
        print  “We should play more Angry Birds”;
    }
```

Statements: while & until

- ‘while’ Example:

```
1.  while  (@names) {
        $name = pop(@names);
        print  “$name\n”;
    }
```

- ‘until’ Example:

```
1.  $count  =  0;
2.  until ($count  == 100)  {
        print  “Keep counting”;
        $count++;
    }
```

Statements: for & foreach

- ‘for’ Example:

```
1.  for ($i = 0;  $i <= $#nums; $i++) {
        print "$nums[$i]\n";
    }
```

- ‘foreach’ Examples:

```
1.  foreach  $digit (@number) {                # $digit refers to the element
        print "$digit\n";
    }
2.  foreach $key (sort keys %agehash) {
        print "$agehash{$key}\n";
    }
```

Statements: last & next

- ‘last’ Example:

```
1.  foreach $key (sort keys %book) {
        if ($book{$key} eq $favorite) {
        print "Found it!";
        last;  }
    }
```

- ‘next’ Examples:

```
1.  foreach $key (sort keys %book) {
        if ($book{$key} eq $boring) {
        next;  }
        print "Read the book!";
        # Code goes here
    }
```

Code Example (1)

```
#!/usr/bin/perl

use Cwd;

my  $base_dir  =  cwd();
my  $home_dir  =  $ENV{"HOME"};

# Read raw data from file and return lines in an array

sub  read_data  {

   my  ($pname,   $sname)  =  @_;
   my  $fname  =  $pname  .  "$sname.txt";

   open  FILE,  "$fname"  or  die  "could not read '$fname'\n";
   my  @lines  =  <FILE>;
   chomp  @lines;
   close  FILE;
   return  @lines;
}
```

Code Example (2)

```
# Print the data separated by $nsep

sub  print_data {
   my  ($outfile, $aref, $osep, $nsep)  =  @_;

   for my $line  (@$aref) {

      # Assuming fields are separated by $osep
      my @fields  =  split  $osep,  $line;
      # Print line
      print  $outfile  join($nsep, @fields),  "\n";
   }
}
```

Code Example (3)

```
# Count the number of categories for an attribute

sub  count_categories {
   my ($aref,  $sep,  $ncol)  =  @_;
   my %categ  =  ();
   my $count  =  0;

   for my $line  (@$aref)  {

      # Assuming fields are separated by $sep
      my @fields  =  split  $sep,  $line;
      my $val  =  $fields[$ncol];
      unless (exists  $categ{$val})  {
         $categ{$val}  =  $count;
         $count++;
      }
   }
   return %categ;
}
```