CS 3101-4 - Programming Languages: Perl

Lecture 1: The basics

Vinodkumar Prabhakaran
vinod@cs.columbia.edu

March 22, 2012

Contents

Course description

- Lectures: Thursdays, 6:10pm-8:00pm, 03/22 - 04/26.
- Location: 834 Seeley W. Mudd Building.
- Instructor: Vinodkumar Prabhakaran
  PhD student, Natural Language Processing
- Office: 850 Inter Church Center, 475 Riverside Dr.
- Office hours: Mon 4pm - 6pm or by appointment.
- Email: vinod@cs.columbia.edu
- Website:
  http://www.cs.columbia.edu/~vinod/coms3101-4.html

Syllabus

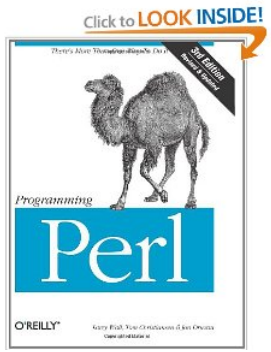
03/22 & 03/29	Basics: scalars, arrays, hashes, strings, lists, contexts, operators, control flow, scopes, I/O, subroutines.
04/05	Regular expressions - matching, substituting, "dirty dozen".
04/12	References, data structures: array of hashes, hash of hashes.
04/19	Object oriented programming in Perl.
04/26	More built in functions, modules, CPAN.

Grading

- Homeworks - 40%
- Project
  - Project proposal - 10%
  - Project submission and report - 40%
- Class participation - 10%

Materials

- Books
  - Programming Perl, 3rd Edition,
    Larry Wall, Tom Christiansen, Jon Orwant
    Recommended, not required.



- Online resources
  - http://perldoc.perl.org/index.html
  - http://www.perlmonks.org/

A brief history

- developed by Larry Wall, a linguist, in 1987
- developed as a general-purpose Unix scripting language to make report processing easier
- Practical Extraction and Report Language
- gained popularity in late 90's as a CGI scripting language
- considered to glue together the internet
- recent versions support graphics, system admin, network programming etc.

Slogans

- "Easy things should be easy and hard things should be possible"
- "There's more than one way to do it"

The genes

- written in C
- derives broadly from C, hence procedural in nature
- multiple paradigms - procedural, object-oriented, and functional styles
- takes many features from shell programming
- lists from Lisp
- hashes from AWK
- regular expressions from sed

Why Perl?

- powerful inbuilt text processing facilities
- modularly extensible - embed perl in other languages; embed other languages in perl.
- often used as a glue language, tying together systems and interfaces that were not specifically designed to interoperate
- data munging - converting or processing large amounts of data
- rapid application development and deployment

Perl in action

- interpreted and dynamic
- mix of compilation and interpretation.
- automatic data-typing
- automatic memory-management
- arrays and strings can grow as they like.

Who uses Perl

- Me.
- cPanel, Slash, Bugzilla, RT, TWiki, and Movable Type.
- Amazon.com, bbc.co.uk, Priceline.com, Craigslist, IMDb, LiveJournal, Slashdot and Ticketmaster.

The community

- Comprehensive Perl Archive Network (CPAN) modules
- Open source development
- Fun with Perl
  - Just Another Perl Hacker
  - Obfuscated Perl Contest
  - Perl Poetry

```
# http://en.wikipedia.org/wiki/Black_Perl
AFTERWORDS: tell nobody.
    wait, wait until time;
    wait until next year, next decade;
        sleep, sleep, die yourself,
        die at last
```

  - Perl Golf

Types of variables

- Scalars
- Arrays
- Hashes

Scalars

- a singular value.
- denoted by $.
- integers, strings, floating-point numbers, references etc.
- no need to declare the exact type of scalar (number vs string)
- perl converts types based on the context

```
1 $age = 42;
2 $pi = 3.14;
3 $earth = 5.97e24;
4 $person = "Vinod";
5 $days[0] = "Sunday";
6 $varref = \$person; # reference to variable $person
```

Arrays

- can hold a set of ordered scalars (lists)
- denoted by @.

```
1 @names = ();  # empty array
2 @names = ("Ben","Jen","Ken");  # initialize array
3 @mixed = (123,"world",3.14);  # different types
4 print $names[1]  # prints Jen. (note use of $)
```

ClassAverage.pl (The C'ish version)

```
1  open (IN,"marks.txt"); # open input file
2  $sum = 0; # $ denotes scalar
3  $count = 0;
4  $line = <IN>; # read one line from file
5  while($line) {  # while a valid line a read
6          @fields = split(' ',$line);
7          $mark = $fields[1];
8          $sum = $sum + $mark;
9          $count++;
10         $line = <IN>; # read another line
11 }
12 $avg = $sum/$count; # floating-point division!
13 print "The average marks of $count students: $avg\n" ;
14 close(IN);
```

Input & Output

marks.txt

```
Jenny 47
Ben 48
John 39
Alex 45
Cindy 42
Bob 28
```

STDOUT

```
The average marks of 6 students: 41.5
```

Basic array operations

Initializing

```
1 @names = ("Ben","Jen","Ken");    # initialize array
2 @names = (Ben,Jen,Ken);    # quotes not needed
3 @ages = (37, 34, 4);
4 @words = qw(a very long sentence);    # quoted words
5 @digits = (0..9);    # range
6 @chars = (a..z);    # range
7 @chars = (ax..bb);    # range (ax, ay, az, ba, bb)
```

Indexing and slicing

```
1 print $names[1], ": ", $ages[1]; # Jen: 34
2 print $names[-1]; # Ken
3 @parents = @names[0,1]; # array slice ("Ben","Jen")
4 @somechars = @chars[1..3,7,9]; # ("b","c","d","h","j")
5 print $#names;  # 2 (last index)
```

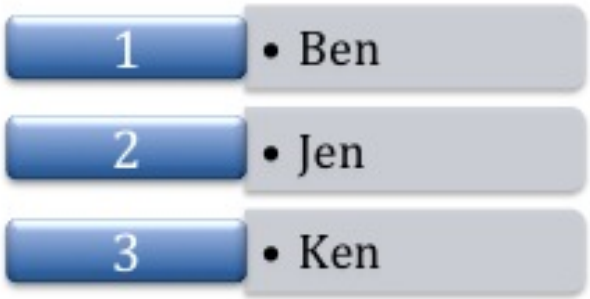
Hashes

- can hold a set of unordered scalars indexes by strings (keys)
- denoted by %

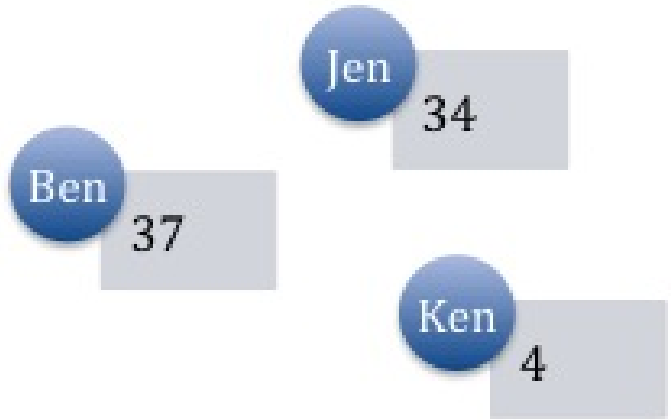
```
1 %ages = (); # empty hash
2 %ages = ("Ben", 37, "Jen", 34, "Ken", 4); # initialize
3 %ages = ("Ben" ==> 37, "Jen" ==> 34, "Ken" ==> 4);
4 print $ages{"Ken"}; # 4
```

Arrays vs Hashes

@names



%ages



Basic hash operations

```
1 %agehash = ("Ben" ==> 37, "Jen" ==> 34, "Ken" ==> 4);
2
3 $age = $agehash{"Ken"}; # $age = 4;
4 @parent_ages = @agehash{"Ben","Jen"}; # hash slice
      (37, 34)
5
6 @names = keys %agehash; # ("Ben", "Ken", "Jen") #
      random order
7 @values = values %agehash; # (37, 4, 34)
8 if (exists $agehash{"Sen"})
```

GradeQuery.pl

```
1  %gradebook = (); # this declaration is not necessary
2  open (IN,"marks.txt");
3  $line = <IN>;
4  while($line) {
5          ($student,$mark) = split(' ',$line);
6          $gradebook{$student} = $mark;
7          $line = <IN>;
8  }
9  while (true) {
10         print "Whose grade? : ";
11         $student = <>;
12         chomp ($student);
13         if ($student eq "q") { last; }
14         if (exists $gradebook{$student}) {
15                 print $gradebook{$student}, "\n";
16         }
17         else {  print "Name does not exist\n";  }
18 }
```

Quotes

- double quotes - interpolation
- single quotes - suppress interpolation
- back quotes - execution

```
1 print "$person is $age years old\n";
2 print 'He owes me $50';
3 `cd /tmp/perl`; # change current dir
```

Interpolation

Disambiguate, if next to alpha-numeric characters

```
1 $verb = "burn";
2 print "${verb}ed"; # burned (not "$verbed")
```

Array interpolation

```
1 print @names; # BenJenKen
2 print "@names";  # Ben Jen Ken. ($" = " ")
```

Escape $ and @ if you want it printed

```
1 $person = "Mr. X";
2 print "$person owes me: \t\$50"; # Mr. X owes me:  $50
```

Context

Scalar context

```
1 $x = mystery();
2 $x[1] = mystery();
3 $x{"key"} = mystery();
```

List context

```
1 @x = mystery();
2 %x = mystery();
```

Scalar context

- Numeric context
- String context
- Boolean context (true or false)
  - a scalar is false, if "", "0" or 0. Is true, otherwise.
  - an undefined value (`undef`) is always false
  - a reference is always true
- Void context

Context examples

```
1 @family = @names; # both list context
2 ($dad) = @family; # both list context, but $dad = "Ben
      "
3 ($dad,$mom) = @family; same as above, $dad = "Ben",
      $mom = "Jen"
4 $num = @family # $num = 3, scalar context
5 print @names; # BenJenKen
6 unlink @files; # deletes all files in the array
7 while(@files) # checks if array is empty, scalar
      boolean context
```

Arrays vs lists

- List is an ordered set of scalars
- Array is a variable that can hold a list

```
1 @names = ("Ben","Jen","Ken");
2 print @names;
3 print ("Ben","Jen","Ken");
4 print "Ben", "Jen", "Ken";
```

```
1 $num = @names; # @names in scalar context, $num = 3.
2 $num = ("Ben","Jen","Ken"); # $num gets "Ken"!!!
```

Operators

Numeric

```
1 $a + $b; # addition
2 $a - $b; # subtraction
3 $a * $b; # multiplication
4 $a / $b; # division (floating point)
5 $a % $b; # modulus / reminder
6 $a ** $b; # exponentiation
```

String

```
1 $a . $b; # concatenation
2 $a x $b; # repeat "baabaabaa"
```

Operators in action

```
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 $screenwidth;
```

More operators

Assignment

```
1 $a = $b + $c;
2 $a = $b = $c = 0;   # All get 0
3 $a += 2;
```

Conditional

```
1 $a = $b ? $c : $b; # similar to C, Java etc
```

Unary arithmetic

```
1 $a++;, ++$a; $a--; --$a; # auto increment/decrement
```

Logical

```
1 $a && $b, $a || $b, !$a
2 $a and $b, $a or $b, not $a, $a xor $b
```

Operators in action

```
1 ($temp -= 32) *= 5/9; # lvalue is returned
2 chomp($query = <>);
3 open (IN, "marks.txt") or die "$!\n";
```

More operators

Comparison

- Numeric: ==, !=, <, >, <=, >=, <=>
- String: eq, ne, lt, gt, le, ge, cmp
- $a <=> $b returns 0 if equal, 1 if $a greater, -1 if $b greater

File Test

- -e (exists?), -r (readable?), -w (writable?)
- -d (directory?), -f (regular file?), -T (text file?)

Operators in action

```
1 if ($query eq "q") { last; }
2 if ($age < 18) { next; }
3 if (-d "marks.txt") { print "this is a dir"; }
```

The if and unless

```
1 $cmp_res = $a <=> $b;
2 if ($cmp_res == 0) { # braces mark blocks. mandatory
3     print "$a is equal to $b";
4 } elsif ($cmp_res < 0) {
5     print "$a is less than $b";
6 } else {
7    print $a is greater than $b";
8 }
```

```
1 unless ($destination eq $home) {
2     print "I am not going home";
3 }
```

The while and until

```
1 while (@files) {
2     $file = pop(@files);
3     print "popped off $file\n";
4 }
```

```
1 $ticket_count = 0;
2 until ($ticket_count == 100) {
3     print "I bought one ticket";
4     $ticket_count++;
5 }
```

More loops: for and foreach

```
1 for ($i = 0; $i <= $#files; $i++) {
2     print "$files[$i]\n";
3 }
```

```
1 foreach $file (@files) {
2     # @files in list context
3     # $file refers to the element, not a copy
4     print "$file\n";
5 }
```

```
1 foreach $key (sort keys %hash) {
2     print "$hash{$key}\n";
3 }
```

Breaking out: last and next

```
1 foreach $key (sort keys %hash) {
2     if ($hash[$key] eq $what_i_am_looking_for) {
3         print "Found what I am looking for";
4         last; # similar to break in C/Java
5     }
6 }
```

```
1 foreach $key (sort keys %hash) {
2     if ($hash[$key] eq $what_i_should_ignore) {
3         next; # similar to continue in C/Java
4     }
5     # do the processing
6 }
```

Special variables $_

```
1 foreach $name (@names) {
2     print $name;
3 }
```

```
1 foreach (@names) {
2     print $_;
3 }
```

```
1 foreach (@names) {
2     print;
3 }
```

Statement modifiers

```
1 take_trash_out() if $you_love_me;
2 shutup() unless $you_want_me_to_leave;
3 feed_my_cat() while $I_am_away;
4 kiss('me') until $I_die;
```

```
1 @fields = split / /, $line;
2 print "field: $_\n" foreach (@fields);
3 print "field: $_\n" foreach split / /, $line;
```

Summary

- scalars, arrays, hashes

```
1 $name = "Ben";
2 @names = (Ben,Jen,Ken);
3 %ages = (Ben ==> 37,Jen ==> 34,Ken ==> 4);
```

- context - scalar vs list

```
1 unlink @files;
2 $num = @files;
```

- quotes and interpolation

```
1 print "Name : $name\n";
2 print "Family : @names";
```

Summary

- operators
  - numeric vs string operators
- control structures
  - if-elsif-else, unless
  - while, until
  - for, foreach
- statement modifiers