COMS 3101
Programming Languages: Perl

Lecture 2

Fall 2013

Instructor: Ilia Vovsha

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

Lecture Outline

- Control Flow (continued)
- Input / Output
- Subroutines
- Concepts:
  - Statement modifiers
  - Special variables
  - More operators
- Array / Hash manipulation
- Next: pattern matching

Remarks

- Perl is case sensitive
- Show warnings: **#!/usr/local/bin/perl -w**
- Block comments: =pod, =cut
  - =pod
  - ….
  - =cut
- Array Size:
  - $arr_size = scalar (@numbers);
  - $str_size = length (“word”);

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
    }
```

Files

- Open / Close a file:
  - Create a filehandle and attach it to a file
  - Predefined filehandles: STDIN, STDOUT, STDERR
  - Specify the behavior you want

- Examples:

```
1.  open (INFILE, “filename”);                  # read (default)
2.  open (OUTFILE, “<$filename”);               # read (explicit)
3.  open (OUTFILE, “>filename”);                # write to a new file
4.  open (OUTFILE, “>>filename”);               # append to existing file
5.  close (INFILE);                  # close file
```

- Write to a file:

```
1.  print  OUTFILE  “some nonsense”;            # write to file
2.  print  STDOUT   “more nonsense”;            # write to STDOUT (explicit)
3.  print  “more nonsense”;                     # write to STDOUT default stream
```

Line Input Operator < … >

- "Angle operator": apply to a filehandle to read the next line
- Examples:
  1. open (INFILE, "in.txt");
  2. open (OUTFILE, ">out.txt");
  3. while ($line = < INFILE >) { # next line
     print OUTFILE "Line:", $line;
  4. close (INFILE, OUTFILE); # close files
- Multiple Lines:
  1. open (INFILE, "in.txt");
  2. $line1 = <INFILE>; $line2 = <INFILE>; # 1st, 2nd lines
  3. @restlines = <INFILE>; # all remaining lines
  4. close (INFILE);

Command Line Arguments

- Can pass arguments into a program from the command line (file names, parameters)
- Stored in the special array @ARGV
- Example:

1. **$ perl test.pl infile.txt outfile.txt 6**
2. If ($#ARGV != 2) {
   print "wrong number!";
   exit ;
   }
   $file1 = $ARGV[0];
   $file2 = $ARGV[1];

Null File Handle

- Empty angle operator: <>
- Loops through @ARGV, each argument mentioned in the command line is assumed to be file
- If no arguments are passed, lines are read from STDIN
- Example:

1. **$ perl test.pl infile.txt**
2. while (<>) { print $_ ; }

3. open (IN, $ARGV[0]);
4. while (<IN>) { print $_ ; }

Subroutines (definition)

- Define anywhere in the program using keyword sub
- Global by default, overloading is implicit (can't have two subs with the same name)
- Calling subroutine requires '&' unless the sub has already been defined (or compiler can tell from the syntax)
- Examples:

1. sub pnext {
       $n++; print "$n";
   }
2. &pnext;

Subs (return values)

- All subs have a return value
- Default return value is the “last calculation performed” (last *expression evaluated* rather than last line of text)
- Example:

```
1.  sub  pnext {
        $n++;  print  “$n”;
    }
2.  $rv  =  &pnext;          # Last expression evaluated is print…
```

- Example:

```
1.  sub  get_list {
        $n++;  ($n..$n+2);
    }
2.  @nums  =  &get_list;          # return list of values
```

Subs (explicit return)

- Example:

```
1. sub   pnext {
             return $n++;
   }
2. $rv  =  &pnext;              # $rv = $n++
```

- Example:

```
1. sub   get_list {
       $n++;  return ($n .. $n+2);
   }
2. @nums  =  &get_list;         # return list of values
```

Subs (passing arguments)

- Use () to pass an argument list (arbitrary # of parameters)
- Parameter list is assigned to @_ (special array, which is local to the sub)
- You can access parameters by: $_[0], $_[1]
- Parameter not used? Throw away
- Parameter not passed? Use as undef
- Examples:

1.
```
sub   max {
        if  ( $_[0]  >  $_[1] ) {
          return $_[0];      }
        else        {
          $_[1];  }
    }
```
2.
```
print  &max(5,88);
```

Subs (private variables)

- Variables are global by default
- Create private variables "*scoped* to the enclosing block" using keyword my
- Examples:

```
1.  sub  max {
        my  $first  =  shift ;   my  $second =  shift ;
        my ($first, $second)  =  @_;
        if  ( $first  >  $second ) {
            return $first;      }
        else        {
            $second;  }
    }
```

Exercise (In Class)

- A sub for computing the minimum element in a list
- Write a subroutine that takes a list of elements and returns the element with minimum value. For example: min(2,33,15,8,99) should return 2.

Concepts

- Statement modifiers
- File test operators
- Special variables

Statement Modifiers

- Simple statements may be followed by a SINGLE modifier before ';':
  - Modifiers: if, unless, while, until, for, foreach
  - Modifier is followed by condition
- Examples:

```
1.  elect_them()  if   $they_make_promises;
2.  curse_them()  unless   $they_keep_promises;
3.  complain()     while      $they_hold_office;
4.  elect_them_again()   until    $they_die;
```

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

File Test Operators

- Test different aspects of a file:
  - -r (readable) -w (writable) -d (directory)
  - -T (text file) -e (exists)
- Examples:
  1. if (-d “code”) { print “this is a dir\n”; }
  2. if (-T “report.txt”) { print “this is a text file”; }
  3. unless (-e $name) { print “no such file”; }

Special Variables

- Predefined variables with a special meaning:
  - Global / local … scalar / array / hash / filehandles
  - $_ : default input/pattern-searching string
  - @_ : subroutine parameters
  - @ARGV : default array for input arguments to script
  - $a, $b : sort comparison routine
- Examples:
  1. `foreach $digit (@number) { print $digit; }`
  2. `foreach $digit (@number) { print $_; }`
  3. `foreach $digit (@number) { print ; }`

Array & Hash Manipulation

- Arrays:
  - Add, remove elements
  - Concatenate, join, split arrays
  - Loop through, sort arrays
- Hashes:
  - Loop through (in sorted order)
  - Check keys
- Related functions: map, grep

Arrays (add, remove)

- End of array: push / pop
- Beginning of array: unshift / shift
- Anywhere: delete
- Examples:

1. @numbers = (4,1,2,5,8,6);
2. push (@numbers, 9);
3. $num = pop (@numbers);
4. $num = shift (@numbers);
5. unshift (@numbers, 9);
6. delete $numbers[1];

Arrays (con, join, split)

- Concatenate arrays: ,
- Join array elements into a string: join
- Split a string into elements of array: split
- Examples:

```
1.  @nums1 = (1..4);      @nums2 = (5..8);
2.  @allnums = (@nums1, @nums2);
3.  print  join (";",  @nums1);
4.  @words = qw (dogs cats pets);
5.  $sentence = join ("_",  @words);
6.  @words = split (/_ /, $sentence);
7.  @words = split ('s', $sentence);
```

Arrays (sort)

- Default order: alphabetical ascending
- Flip array: reverse
- Examples:

```
1.  @numbers  =  (11,13,2);
2.  @names = (SF, LA, NY, Z);
3.  @sorted  =  sort (@names);                  # order: alpha asc
4.  @sorted  =  sort (@numbers);                # order: alpha asc (WRONG!)
5.  @sorted  =  sort  {$a  < = >  $b} (@numbers);          # numeric asc
6.  @sorted  =  sort  {$b  < = >  $a} (@numbers);          # numeric desc
7.  @sorted  =  sort  {$b  cmp  $a} (@names);               # alpha desc
8.  @reversed  =  reverse (@numbers);                       # flip
```

Arrays (loop through)

- Example:

```
1. foreach   $digit  (@number)  {
       # process $digit
       # any changes to $digit carry over to the array
       # @number
   }
```

- Example:

```
1. foreach   $key  (keys  %h1)  {
       # processing $key, which may affect hash (h1)
       $h1{$key}++;
   }
```

Hashes (loop through)

- Keys in order:

```
1. foreach  $key  (sort  keys  %h1)  {
       print   “$key:  $h1{$key} \n”;
   }
```

- Values in (descending) order:

```
1. foreach  $key  (sort  {$h1{$b} <=> $h1{$a}}  keys  %h1)
   {
       print   “$key:  $h1{$key} \n”;
   }
```

Hashes (size, delete, reverse)

- Number of pairs: scalar
- Remove entry: delete
- Swap keys and values: reverse (values must be unique!)
- Examples:

```
1.  %h1 = (a=>1, b=>2, c=>2);
2.  $num = scalar ( keys %h1 );
3.  delete $h1{“aa”};                # delete key “aa”
4.  %h2 = reverse (%h1);             # swap keys & values (WRONG!)
```

Hashes (check keys)

- Does the key exist? exists
- Is the value defined? defined
- Examples:

```
1.  %hs  =  (a, 1, b);          # a => 1, b => undef
2.  If  ( exists  $hs{a}  ) {   # key “a” exists
3.  If  ( defined  $hs{a}  ) {  # value for key “a” is defined (true)
4.  If  ( defined  $hs{b}  ) {  # value for key “b” is undef (false)
```

map & grep

- Transform list element-wise: map
  - @new_list = map { CODE } @old_list;
  - CODE is applied to an element of old_list, return value is stored in new_list
- Filter list element-wise: grep
  - @new_list = grep { CODE } @old_list;
  - CODE is executed for each element of old_list, if code returns true, element is placed in new_list
- Examples:

```
1. @n2 = map { $_ * $_ } @numbers;        # square each element
2. @names2 = map { lc } @names;           # lower case each element
3. @n2 = map { $_ * $_ } grep {$_ > 5} @numbers;
   #  square each element if > 5.
   #  'map' returns different number of elements!
```

Exercise (In Class)

- Compute the average grade for a list of students
- Suppose you have a file with a list of {student,grade} pairs (each pair on a separate line separated by a comma). Write a program that receives the file-name as a command line argument, computes the average grade for the class and prints the result to the screen.