COMS 3101
Programming Languages: Perl

Lecture 3

Fall 2013

Instructor: Ilia Vovsha

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

Lecture Outline

- Array / Hash manipulation (continued)
- Pattern Matching (examples and rules)
- Concepts:
  - More on files
  - Scoped variables
  - Special variables
- Next: References
- Next: Data Structures

Remarks

- “lvalue”: left-value, left side of an assignment, storage location (address) you can assign a new value to
- Conditional operator “?” (if-then-else):
  - EXP1 ? EXP2 : EXP3;
  - If EXP1 == TRUE, then EXP2, else EXP3
- Functions:
  - unlink @Files; # Delete all files in array
  - $str = lc ($STR); # lower case expression
  - @items = splice(@arr,2,3) # Remove from middle of array

Exercise Solution (min sub)

```
sub  min {
    my  $cur_min  =  shift ;
    foreach  (@_)  {
        if  ( $_  <  $cur_min )  {
          $cur_min  =  $_ ;
        }
    }
    return $cur_min;
}
```

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 (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
```

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.

Exercise Solution (average grade)

```
open (IN, $ARGV[0]);
@lines  = <IN>;
$sum  =  $count  =  0;
foreach   $line  (@lines) {
     ($student,  $mark)  =  split (/ /, $line) ;
     $sum += $mark;
     $count++;
}
$avg  =  $sum / $count;
close (IN);
```

Pattern Matching

- Regular Expressions (REGEX)
- Operators (cages): match, substitute
- Elements:
  - Metacharacters
  - Pattern modifiers
  - Character classes (classic, Unicode)
  - Quantifiers
  - Assertions
  - Grouping & capturing
  - Alternation
  - Magic dot
  - Return values
- General advice

Pattern Matching (purpose)

- MS degree requirement

> Fulfill the 12-credit core requirement. One of the core requirements must be CSOR W4231. In addition, COMS W3261 or past equivalent is a required pre-requisite (No MS credit for W3261).
>
> 1 required course: COMS W4236.
>
> 1 course chosen from the "Electives I" list: COMS W4203, COMS W4205, COMS W4241, COMS W4252, COMS W4261, or COMS W4281.

- Task: list all course mentions
- Solution: array of all numbers and check if its in each line?

```
@allcourses  =  ("COMS W3000",...,"CSOR W3000",...);
while (<>)   {
 foreach  $x  (@allcourses)    {
    if  (index ($_, $x) > 0)   {                    # returns index of $x in $_
         print  $x,  ",  ";       }    }
}
```

Pattern Matching (regex)

- Objective: search large data efficiently, extract useful info
- PERL is “Practical Extraction & Report Language”
- Approach: create a language to specify patterns
  - Engine to determine if pattern matches data
  - REGEX: easy to understand unlike long code
- General language which can be extended to accommodate new characters & patterns

Pattern Matching (match, substitute)

- PM operators: “cages” for regex
- Match: m/PATTERN/ (m optional)
- Substitute: s/PATTERN/REPLACEMENT
- Bind match/substitute to a variable:
  - $line =~ m/PATTERN/
  - $line !~ m/PATTERN/
- Two-pass parsing: interpolate before interpreting regex
- Inside cage: power of “” Interpolation

Pattern Matching Example

Fulfill the 12-credit core requirement. One of the core requirements must be CSOR W4231. In addition, COMS W3261 or past equivalent is a required pre-requisite (No MS credit for W3261).

1 required course: COMS W4236.

1 course chosen from the "Electives I" list: COMS W4203, COMS W4205, COMS W4241, COMS W4252, COMS W4261, or COMS W4281.

- Task: list all course mentions
- Solution: match a pattern

```
while (<>)  {
  if  ($_  =~  m/PATTERN/)  {
        print  $&,  ",  ";        }                    # $& contains the matched part
}
```

PM Example (character class)

Fulfill the 12-credit core requirement. One of the core requirements must be CSOR W4231. In addition, COMS W3261 or past equivalent is a required pre-requisite (No MS credit for W3261).

1 required course: COMS W4236.

1 course chosen from the "Electives I" list: COMS W4203, COMS W4205, COMS W4241, COMS W4252, COMS W4261, or COMS W4281.

- Match target: any digit
- Output: COMS W3261, COMS W4236, COMS W4203,

```
while (<>)  {
  if  ($_  =~  m/COMS  W\d\d\d\d/)  {
        print  $&,  ",  ";        }                    # $& contains the matched part
}
```

PM Example (quantifier)

Fulfill the 12-credit core requirement. One of the core requirements must be CSOR W4231. In addition, COMS W3261 or past equivalent is a required pre-requisite (No MS credit for W3261).

1 required course: COMS W4236.

1 course chosen from the "Electives I" list: COMS W4203, COMS W4205, COMS W4241, COMS W4252, COMS W4261, or COMS W4281.

- Match target: four digits
- Output: COMS W3261, COMS W4236, COMS W4203,

```
while (<>)  {
  if  ($_  =~  m/COMS  W\d{4}/)  {
        print  $&,  ",  ";              }
}
```

PM Example (alternation)

Fulfill the 12-credit core requirement. One of the core requirements must be CSOR W4231. In addition, COMS W3261 or past equivalent is a required pre-requisite (No MS credit for W3261).

1 required course: COMS W4236.

1 course chosen from the "Electives I" list: COMS W4203, COMS W4205, COMS W4241, COMS W4252, COMS W4261, or COMS W4281.

- Match target: one of two strings
- Output: CSOR 4231, COMS W4236, COMS W4203,

```
while (<>)  {
  if  ($_  =~  m/(COMS|CSOR)  W\d{4}/)  {
        print  $&,  ",  ";                    }
}
```

PM Example (modifier)

Fulfill the 12-credit core requirement. One of the core requirements must be CSOR W4231. In addition, COMS W3261 or past equivalent is a required pre-requisite (No MS credit for W3261).

1 required course: COMS W4236.

1 course chosen from the "Electives I" list: COMS W4203, COMS W4205, COMS W4241, COMS W4252, COMS W4261, or COMS W4281.

- Match target: all patterns
- Output: CSOR 4231, COMS W3261 , COMS W4236, COMS W4203, COMS W4205, COMS W4241, COMS W4252, COMS W4261, COMS W4281,

```
while (<>)  {
  while ($_  =~  m/(COMS|CSOR)  W\d{4}/g)   {
        print  $&,  ",  ";                                  }
}
```

PM Example (character class)

Fulfill the 12-credit core requirement. One of the core requirements must be CSOR W4231. In addition, COMS W3261 or past equivalent is a required pre-requisite (No MS credit for W3261).

1 required course: COMS W4236.

1 course chosen from the "Electives I" list: COMS W4203, COMS W4205, COMS W4241, COMS W4252, COMS W4261, or COMS W4281.

- Match target: all patterns (four letter word)
- Output: CSOR 4231, COMS W3261 , COMS W4236, COMS W4203, COMS W4205, COMS W4241, COMS W4252, COMS W4261, COMS W4281,

```
while (<>)  {
  while ($_  =~  m/[A-Z]{4} [A-Z]\d{4}/g)  {
        print  $&, ", ";                    }
}
```

PM Example (grouping / capturing)

Fulfill the 12-credit core requirement. One of the core requirements must be CSOR W4231. In addition, COMS W3261 or past equivalent is a required pre-requisite (No MS credit for W3261).

1 required course: COMS W4236.

1 course chosen from the "Electives I" list: COMS W4203, COMS W4205, COMS W4241, COMS W4252, COMS W4261, or COMS W4281.

- Match target: all patterns + capture the pattern
- Output: CSOR 4231, COMS W3261 , COMS W4236, COMS W4203, COMS W4205, COMS W4241, COMS W4252, COMS W4261, COMS W4281,

```
while (<>)  {
  while ($_  =~  m/([A-Z]{4} [A-Z]\d{4})/g)  {
        print  $1, ", ";                          }
}
```

PM Example (more grouping)

Fulfill the 12-credit core requirement. One of the core requirements must be CSOR W4231. In addition, COMS W3261 or past equivalent is a required pre-requisite (No MS credit for W3261).

1 required course: COMS W4236.

1 course chosen from the "Electives I" list: COMS W4203, COMS W4205, COMS W4241, COMS W4252, COMS W4261, or COMS W4281.

- Match target: all patterns + capture part of the pattern
- Output: 4231, 3261 , 4236, 4203, 4205, 4241, 4252, 4261, 4281,

```
while (<>)   {
  while ($_   =~   m/([A-Z]{4}) ([A-Z])(\d{4})/g)    {
        print   $3,  ",  ";                                    }
}
```

Pattern Matching

- Elements:
  - Metacharacters
  - Pattern modifiers
  - Character classes (classic, Unicode)
  - Quantifiers
  - Assertions
  - Grouping & capturing
  - Alternation
  - Magic dot
  - Return values

PM (metacharacters)

- Same “Dirty Dozen” characters
- Longer sequences: metasymbols
- Backslash \ : escapes MC
- Types of metasymbols:
  - General
  - Quantifiers
  - Alphanumeric
  - Wildcard
  - Specific, Extended

- \ - next character has special meaning
- | - alternator
- (- grouping
-) - grouping
- [- character class
- { - quantifier
- ^ - match at the beginning
- $ - match at the end
- * - zero or more times
- + - one or more times
- ? - optional (zero or one time)
- . - match anything

PM (modifiers)

- Modifier (flag, option): following final delimiter
  - /i ignore case
  - /g globally find all matches
  - /x improve legibility (permit ws, comments)
  - /cg continue search after (global) match failed
- Examples:
  1. if ($line =~ m/CS \d{4}/i) { … }
  2. while ($line =~ /CS \d{4}/gi) { … }
  3. if ($line =~ /CS \w \d{4}/x) { … }

PM (character classes)

- Match ONE character with (without) particular property
  - *Custom*: list of characters inside [ ]
  - *Classic*: shortcuts for common classes \d \D \w \W \s \S
  - *Unicode*: industry standard for representation \p{PROP}
- Rules:
  - Specify range with '-': [a-z], [A-D0-9]
  - Invert class with '^': [^abc] [a-d^e]
  - Upper-case shortcut inverts classic class: [^0-9] \D ^\d
  - Metasymbols lose meaning
- Examples:
  1. if ($line =~ /[a-z]\D \d{1,3}/) # lc letter, nondigit, 1-3 digits
  2. if ($line =~ /ab\t[ab]\W\s/) # ab,tab,(a or b),nonword,ws

PM (quantifiers)

- How many times something may match (once by default)
- Effectively loops in regex programs
- Greedy matching by default (match as many as possible)
- Minimal instead of maximal (greedy) by placing a ? after the quantifier
- Leftmost matching (the earliest match wins)
- Rules:
  - {MIN,MAX} {MIN,} {COUNT} "at least-at most….at least….exactly"
  - * + ? : "0 or more….1 or more….0 or 1 time"
  - * + ? equivalent to {0,} {1,} {0,1}

Quantifiers (examples)

- Examples:

```
1. if ($line =~ /abc/)          # abc
2. if ($line =~ /abc*/)         # ab, abccc, but not abcabc
3. if ($line =~ /abc+/)         # abccc, but not ab or abcabc
4. if ($line =~ /(abc)+/)       # abc, abcabc etc.
5. if ($line =~ /abc?/)         # ab and abc
6. if ($line =~ /abc{2}/)       # abcc and nothing else
7. if ($line =~ /(abc){2}/)     # abcabc and nothing else
8. if ($line =~ /abc{2,}/)      # abcc, abccc and so on
9. if ($line =~ /abc{2,4}/)     # abcc, abccc or abcccc
```

PM (assertions)

- Positions in the string to be matched (tie pattern to position)
- Do not correspond to a character ("zero-width")
  - ^ beginning of string (line if using 'm' modifier)
  - $ end of string (line if using 'm' modifier)
  - \b any word boundary (\w\W or \W\w)
  - \B NOT a word boundary
- Examples:

```
1. if ($line =~ /^S\d{4}/)  # line starts with S followed by digits
2. if ($line =~ /\d$/)      # line ends with digit
3. if ($line =~ /\bis\b/)   # "what is it" but NOT "artist"
```

PM (grouping & capturing)

- Group and remember sub-patterns using ( )
- After the pattern: $N denotes the Nth group from start
- Within the pattern: \N denotes the Nth group from start
- Dynamic scope until end of block or next match
- Can use them with s/// as replacement
- Nested (()): counting by location of left parenthesis '('
- Special variables:
  - $` all to left of match
  - $& entire matched string
  - $' all to right of match

Grouping & Capturing (examples)

- Examples:

```
1.  if  ($line =~ /<(.*?) >.*?<\/\1>/)
        # matches any tags <xyz>.....</xyz>

1.  if  ($line =~ /<(.*?) >.*?<\/\1>/) {
2.          print  "$1  :  $2";                    }

1.  $line  =~  s/^(\w+) , (\w+)/$2 , $1/;        # swap two words
2.  $line  =~  s/((\w+) , (\w+))/;               # nested (())
3.  $line   =  "perl is fun , or is it";
4.  $line  =~ m/((\w+) , (\w+))/;
5.  print  "pre - match – post: $`\t$&\t$'\n";
    # prints "….perl is    fun , or        is it"
```

PM (alternation)

- Specify a set of possibilities (|)
- For overlapping matches, place longer string first
- Extends only to the innermost enclosing ()
- Examples:
  1. if ($line =~ /bank|banker/) # always matches bank!
  2. if ($line =~ /(banker|bank)/) # banker can match
  3. if ($line =~ /ban(k|b)er/) # banker or banber

PM (magic .)

- The dot . matches any character (wildcard)
- Turn maximal (greedy) matching into minimal matching by adding a '?'
- Examples:

1. if ($line =~ /<xml>(.*)<\/xml>/)
   # match any text between two xml tags

1. $line = "Want this <bold>content's tag<bold> matched ";
2. if ($line =~ /<(.*)>/) # matches "bold>content's tag <bold"
3. if ($line =~ /<(.*?)>/) # matches "bold"
4. if ($line =~ /<(.+?)>/) # matches only if its non empty

PM (return values)

- Match (m//) :
  - scalar context - returns true (1) if successful, else false (“”)
  - list context - returns a list of matched groups
- Substitute (s///):
  - returns number of times it succeeded (scalar & list context)

```
$line  =  "<text>this is the text</text>";
($tag, $content)  =  $line  =~  /^<(.*?)>(.*?)<\/\1>$/;

if  (@perls  =  $text  =~  /perl/gi)  {
   print  "Number of times Perl mentioned : ",  scalar(@perls);  }

$string  =  "name=xyzzy id=9 score=0";
%hash  =  $string  =~  /(\w+)=(\w+)/g;

$num  =  $text  =~  s/perl/PERL/g;
```

PM (general advice)

- When matching multiple regex, list common case first
- When writing a long regex, simplify with variables (interpolation)
- Consider using || (logical) instead of | (regex) to be more efficient
- Avoid $& $` $’ if you can (slows down execution). However, if used once, use all the time without penalty
- Not every problem should be solved with regex: consider functions to manipulate strings (substr) instead
- Start by writing down all the patterns you need to identify, then proceed to contrive the regex

More Examples

```
while (<>)  {
  next  if  $line  =~  /^#/;
  …
}
```

```
from:      vp2198@columbia.edu
to:   vp2198@columbia.edu
date:      Sun, Apr 1, 2012 at 11:08 PM
subject:   COMS W3101.004 Office Hours Change (this week only)
mailed -by:     columbia.edu

while (<>)  {
      $_  =~  /^(.*?):\s*(.*)$/;
      $hash{$1}  =  $2;
}
```

Concepts

- More on files
- Scoped variables
- Special variables

< … > Rules

- "Angle operator": apply to a filehandle to read the next line
- Auto assignment to $_ **only** in while loop! (not if, unless,…)
- Examples:

1.
```
while ( < INFILE > ) {                    # next line
    print $_;
```

2.
```
if ( < INFILE > ) {                       # WRONG
    print $_;                             # prints whatever in $_ before
```

3.
```
while ( <IN1> && <IN2> ) {                # WRONG (throw away lines)
    print $_;                             # prints whatever in $_ before
```

Scoped Variables

- my
  - creates private variable visible only within block
  - hidden from outside of enclosing scope, and hides previously declared variables with identical name
  - confines *name & value* to scope
  - suitable for scalar/array/hash variables
- our
  - confines *name* only to scope (no effect on visibility)
  - suitable for scalar/array/hash variables
  - used to access global variables, their initial value inside block unchanged
  - effects or assignment persist after the scope of declaration (block)
- local
  - confines *value* only to scope
  - suitable for scalar/array/hash + more variables
  - initial value for variable is () or undef
  - value of variable is restored no matter how you exit the block (changes thrown away)
  - “dynamic” scope: value of variable depends on scope & changes during run-time
  - ‘my’ is preferable over ‘local’

Scoped Variables (example)

```
$office  =  "global";                    # Global $office
&say();                                  # prints "global"
&barney();                               # prints "barney global", lexical scope;
&fred();                                 # prints "fred fred", dynamic scope,
&say();                                  # prints "global", restored after &fred()

sub  say  { print  "$office\n";  }        # print the $office

sub  barney  {
       my  $office  =  "barney";
       print  "$office ";  &say();  }

sub  fred  {
       local $office = "fred";
       print "$office ";  &say();  }
```

Special Variables

- Predefined variables with a special meaning:
  - $_ @_ @ARGV $a $b
  - $1,$2,… : matched groups in pattern
  - $&, $`, $’ : match, pre-match, post-match pattern
  - $0 : program/script file name