More Regular Expressions

List/Scalar Context for m//

- Last week, we said that m// returns ‘true’ or ‘false’ in scalar context. (really, 1 or 0).
- In list context, returns list of all matches enclosed in the capturing parentheses.
  - $1, $2, $3, etc are still set
- If no capturing parenthesis, returns (1)
- If m// doesn’t match, returns ( )

Modifiers

- following final delimiter, you can place one or more special characters. Each one modifies the regular expression and/or the matching operator
- full list of modifiers on pages 150 (for m//) and 153 (for s///)

/i Modifier

- /i ➔ case insensitive matching.
- Ordinarily, m/hello/ would not match “Hello.”
  - we saw this last week with the banana example
- However, this match *does* work:
  - print “Yes!” if (“Hello” =~ m/hello/i);
- Works for both m// and s///

/s Modifier

- /s ➔Treat string as a single line
- Ordinarily, the . wildcard matches any character *except* the newline
- If /s modifier provided, Perl will treat your regexp as a single line, and therefore the . wildcard will match \n characters as well.
- Also works for both m// and s///

/m Modifier

- /m ➔ Treat string as containing multiple lines
- As we saw last week, ^ and $ match “beginning of string” and “end of string” only.
- if /m provided, ^ will also match right after a \n, and $ will match right before a \n
- Yet again, works on both m// and s///

/o Modifier

- /o ➔ Compile pattern only once
- Ordinarily, a pattern containing a variable is sent through variable interpolation engine every time matching operation evaluated
  - (unless delimiters are single quotes, of course)
- with /o modifier, variable is interpolated only once
- if variable changes before next time pattern match is done, Perl doesn't notice (or care) – it still evaluates original value of the variable
- Yes, both m// and s/// again

/x Modifier

- /x ➔ Allow formatting of pattern match
- Ordinarily, whitespace (tabs, newlines, spaces) inside of a regular expression will match themselves.
- with /x, you can use whitespace to format the pattern match to look better
- m/\w+:(\w+):\d{3}/;
  - match a word, colon, word, colon, 3 digits
- m/\w+ : (\w+) : \d{3}/;
  - match word, space, colon, space, word, space, colon, space, 3 digits
- m/\w+ : (\w+) : \d{3}/x;
  - match a word, colon, word, colon, 3 digits

More /x Fun

- /x also allows you to place comments in your regexp
- Comment extends from # to end of line, just as normal

```
m/          #begin match
  \w+ :     #word, then colon
  (\w+)     #word, returned by $1
  : \d{3}   #colon, and 3 digits
/x          #end match
```

- Do not put end-delimiter in your comment
- yes, works on m// and s/// (last one, I promise)

/g Modifier (for m//)

- List context:
- return list of all matches within string, rather than just ‘true’
  - if capturing parentheses, return all occurrences of those sub-matches
  - if not, return all occurrences of entire match

```
$nums = “1-518-276-6505”;
@nums = $nums =~ m/\d+/g;
   # @nums ➔ (1, 518, 276, 6505)
$string = “ABC123 DEF GHI789”;
@foo = $string =~ /([A-Z]+)\d+/g;
   # @foo ➔ (ABC, GHI)
```

More m//g

- Scalar context:
- initiate a ‘progressive’ match
- Perl will remember where your last match on this variable left off, and continue from there

```
$s = “abc def ghi”;
for (1..3){
  print “$1” if $s =~ /(\w+)/;
} #prints abcabcabc
for (1..3){
  print “$1” if $s =~ /(\w+)/g;
} #prints abcdefghi
```

/c Modifier (for m//)

- Used only in conjunction with /g
- /c ➔ continue progressive match
- When m//g finally fails, if /c used, don’t reset position pointer

```
$s = “Billy Bob Daisy”;
while ($s =~ /(B\w+)/g){ print “$1 ”; }
#prints Billy Bob
print $1 if ($s =~ /(\w+i\w+)/g);#prints Billy

while ($s =~ /(B\w+)/gc){ print “$1 ”; }
#prints Billy Bob
print $1 if ($s =~ /(\w+i\w+)/g);#prints Daisy
```

/g Modifier (for s///)

- /g ➔ global replacement
- Ordinarily, only replaces first instance of PATTERN with REPLACEMENT
- with /g, replace all instances at once.

```
$a = ‘$a / has / many / slashes /’;
$a =~ s#/#\\#g;
```

- $a now ➔ ‘$a \ has \ many \ slashes \’

Return Value of s///

- Regardless of context, s/// always returns the number of times it successfully search-and-replaced
- If search fails, didn’t succeed at all, so returns 0, which is equivalent to false
- unless s///g modifier is used, will always return 0 or 1.
- with /g, returns total number of global search-and-replaces it did

/e Modifier

- /e ➔ Evaluate Perl code in replacement
- Looks at REPLACEMENT string and evaluates it as perl code first, then does the substitution

```
s/
hello
/
“Good ”.($time<12?“Morning”:“Evening”)
/xe
```

A Bit More on Clustering

- So far, we know that after a pattern match, $1, $2, etc contain sub-matches.
- What if we want to use the sub-matches while still in the pattern match?
- If in replacement part of s///, no problem – go ahead and use them:
- `s/(\w+) (\w+)/$2 $1/; # swap two words`
- if still in match, however….

Clustering Within Pattern

- to find another copy of something you've already matched, cannot use $1, $2, etc…
  - operation passed to variable interpolation *first*, then to regexp parser
- instead, use \1, \2, \3, etc…
- `m/(\w+) \1/;  #find duplicate words`

Transliteration Operator

- tr/// ➔ does not use regular expressions.
  - Probably shouldn't be in RegExp section of book
  - Authors couldn't find a better place for it.
    - Neither can I
- tr/// *does*, however, use binding operators =~ and !~
- formally:
- tr/SEARCHLIST/REPLACEMENTLIST/;
  - search for characters in SEARCHLIST, replace with equivalent characters in REPLACEMENTLIST

What to Search, What to Replace?

- Much like character classes (from last week), tr/// takes a list or range of characters.
- tr/a-z/A-Z/;
  - replace any lowercase characters with equivalent capital character.
- TAKE NOTE: SearchList and ReplacementList are NOT REGULAR EXPRESSIONS
  - attempting to use RegExps here will give you errors
- Also, no variable interpolation is done in either list

tr/// Modifiers

- /c ➔ Compliment searchlist
  - 'real' search list contains all characters *not* in given searchlist
- /d ➔ Delete found but un-replaced characters
  - tr/a-z/A-N/d; #replace a-n with A-N. Delete o-z.
- /s ➔ Squash duplicate replaced characters
  - sequences of characters replaced by same character are 'squashed' do single instance of character

tr/// Notes

- if Replacement list is shorter than Search list, final character repeated until it's long enough
  - `tr/a-z/A-N/;`
  - `#replace a-m with A-M.`
  - `#replace n-z with N`
- if Replacement list is null, repeat Search list
  - useful to count characters, or squash with /s
- if Search list shorter than Replacement list, ignore 'extra' characters is Replacement
- if no binding string given, tr/// operates on $_, just like m// and s///