Programming in Perl
Lecture 4 Examples

Read-Only Variables used with Regexes

Name	Perl Variable	Holds copy of
Match	$&	substring matched in string
Prematch	$‘	portion of string before matched substring
Postmatch	$’	portion of string after matched substring

Note that these variables are modified each time a regex search is done.

```
$_ = "baaad";

if (/a{4}/) {
  print("$&\n");
  print("$‘\n");
  print("$’\n");
}
# nothing is printed; pattern
# doesn’t match any portion
# of the string
```

```
$_ = "baaaaad";

if (/a{4}/) {
  print("$&\n");
  print("$‘\n");
  print("$’\n");
}
# prints:
# aaaa<newline>
# b<newline>
# ad<newline>
```

```
$_ = "baaaad";

if (/[^a]*a{4}[^a]*/) {
  print("$&\n");
  print("$‘\n");
  print("$’\n");
}
# prints:
# baaaad<newline>
# <newline>
# <newline>
```

```
$_ = "baaaad";

if (/a{4}/) {
  print("$&\n");
  print("$‘\n");
  print("$’\n");
}
# prints:
# aaaa<newline>
# b<newline>
# d<newline>
```

```
$_ = "baaaaad";

if (/[^a]*a{4}[^a]*/) {
  print("$&\n");
  print("$‘\n");
  print("$’\n");
}
# prints nothing
```

( ) vs (?:):

```
$_ = "apples pears peaches plums";
/(?:\w*)\s(\w*)\s(\w*)/;
# matches 0 or more word characters followed by
# 0 or more word characters, etc.
print ("$1\n$2\n$3\n");
# $1 is assigned to by the first (),
# not the (?:); prints
Use of uninitialized value at ./parens.pl line 7.
pears<newline>
peaches<newline>
<newline>
```

Anchors:

```
Anchor Pattern      Meaning
--------------      -------
^                   Matches only at beginning of string
$                   Matches only at end of string
\b                  Matches at a word boundary
                    (between characters that match \w and \W)
\B                  Matches except at a word boundary
(?=regex)           Matches if engine would match <regex> next
(?!regex)           Matches if engine wouldn't match <regex> next


Examples:

 /^Al/;  # matches Al iff Al is at start of string
          # "Al said hi" match
          # "Hi Al" no match
 /Al$/;  # matches Al iff Al is at end of string
          # "Al said hi" no match
          # "Hi Al" match "Hi Al\n" match as well

 # word boundaries ; note : hello_there is a word
 #                          9.9 are two different words

 /Al\b/;  # "Al said hi" match
          # "Albert said hi" no match
 /\bAl/;  # "Hi Allen" match
          # "Mr. vanAllen" no match
 /\bAl\b/; # "Al" match
           # "Albert said hi" no match
           # "Mr. vanAllen" no match
 /\bAl\B/; # "Albert said hi" match
           # "Al said hi" no match

 # last two called lookahead anchors

 /Bill (?=The Cat|Gates)/; # matches only strings in which
                           # "The Cat" or "Gates" comes
                           # after "Bill "
```

```
/Bill (?!The Cat|Gates)/; # matches only strings in which
                         # neither "The Cat" nor "Gates"
                         # comes after "Bill "
```

Precedence:

```
regex Grouping Precedence

Name                    Representation
----                    --------------
Parentheses             ( ) (?: )
Multipliers             ? + * {m,n} ?? +? *? {m,n}?
Sequence & anchoring    abc ^ $ (?= ) (?! )
Alternation             |

Examples of use of parentheses:

hi*                 # matches "h", "hi", "hii", "hiii", ...
                    # within a string
(hi)*               # matches "", "hi", "hihi", ...
                    # within a string

/^fee|fie|foe$/;    # matches "fee" at beginning of the string
                    # "fie" anywhere OR "foe" at the end of the
                    # string
/^(fee|fie|foe)$/;  # matches a string consisting solely of
                    # "fee" , "fie" or "foe"

/to(nite|night)/    # matches 'tonite' or 'tonight'
/toni(te|ght)/      # same as above; more efficient
```

Pattern Binding Operator (=~):

```
$name = "Joe Smith";
if ($name =~ /Ren|Stimpy/) {
  print ("Goodnight $&\n");
}

# can be used on anything that yields a scalar value
do {
  # stuff
  print("Continue (y/n)?");
} until (<STDIN> =~ /^[nN]/);

-- ignoring case (case insensitivity)

do {
  # stuff
  print("Continue (y/n)?");
} until (<STDIN> =~ /^n/i);
```

Substitution:

```
        # basic format:
        s/regex-old-string/regex-new-string/

        $_ = "foot fool buffoon";
        s/foo/bar/;
        # $_ is now "bart fool buffon"
        # changes first match encountered

        $_ = "foot fool buffoon";
        s/foo/bar/g;
        # $_ is now "bart barl bufbarn"
        # changes all matches encountered (global)

        $_ = "foOt Fool buffOon";
        s/foo/bar/gi;
        # $_ is now "bart barl bufbarn"
        # changes all matches encountered (global)
        # case insensitive

        $_ = "hello world";
        $new = "goodbye";
        s/hello/$new/;
        # replaces hello with goodbye

# use of x to allow commenting
$number = 9999999999;
$number =~ s/
    (\d{1,3})        # before a comma: one to three digits
    (?=              # followed by, but not part of what's matched
       (?:\d\d\d)+   #    some number of triplets...
       (?!\d)        #    ...not followed by another digit
    )                # (which ends the number)
    /$1,/gx;         # x allows regular expression to be broken
                     # across lines and comments to be inserted
print("$number\n");
# the result printed is 9,999,999,999
```

Variable interpolation and regexes:

```
      # open "file.dat" for reading, associating
      # the file with filehandle INFILE
      open(INFILE, "file.dat");
      # open "matches.out" for writing, associating
      # the file with filehandle OUTFILE
      open(OUTFILE, ">matches.out");
      print("Enter a word to search for: ");
      chomp($search_str = <STDIN>);

      # continued next page ...
```

```
      # search through file for $search_str
      # as if it were a single word
      while(defined($line = <INFILE>)) {
        chomp($line);
        if ($line =~ /\b$search_str\b/) {
          print OUTFILE ("$line\n");
        }
      }

      close(INFILE);
      close(OUTFILE);


        -- Suppose file.dat is
toy boy
roy camel llama
```

Use of Quote Escape to Backslash Escape Regex Pattern Characters:

```
        -- \Q : the quote escape

      $what = "[box]";
      foreach (qw (in[box] out[box] white[sox])) {
        if (/\Q$what\E/) {
          print ("$_ matched!\n");
        }
      }
      # matches [box] of 'in[box]' and
      # 'out[box]' but not [sox] of 'white[sox]'
```

Example regex (recognizes complex numbers) :

```
$digit = '\d';
$digits = "$digit+";
# omitting definition of float
$int = "[+-]?$digit+";
$real = "(?:$float|$int)";
$imag = "(?:${real})?i";
$opt_spaces = '\s*';
# qq double quotes everything between < and >;
# this is always treated as a single string
$complex = qq<
    $real            # real part
    $opt_spaces      # 0 or more spaces
    [+-]             # + or -
    $opt_spaces      # 0 or more spaces
    $imag            # imaginary part
>;

# strip comments out of string
$complex =~ s/#.*//g;
# strip spaces and newlines out of string
$complex =~ s/\s+//g;
```

Split:

```
$line = "Betty Boop:555-5555:1 Boop Lane::100000";
# split line use : as delimiter
@fields = split(/:/,$line);
# @fields is ("Betty Boop", "555-5555", "1 Boop Lane",
#             "", "100000")

@fields = split(/:+/,$line);
# @fields is ("Betty Boop", "555-5555", "1 Boop Lane",
#             "100000")

# empty trailing fields are ignored
$line = "Betty Boop:555-5555:1 Boop Lane:100000:";
# split line use : as delimiter
($name,$phone,$address,$salary,$dob) = split(/:/, $line);
# $dob is undef
```

Join:

```
# @fields is ("Betty Boop", "555-5555", "1 Boop Lane",
#             "100000")
# put $line back together
$gluedline = join(":", @fields);
# "Betty Boop:555-5555:1 Boop Lane:100000"

# note: the glue string is just a string NOT a regex

# to get glue in front of a list as well
$result = join("+", "", @fields);
# "" is treated as the empty element to be glued
# with the first data element of @fields
# $result is "+Betty Boop+555-5555+1 Boop Lane+100000";

# to get glue in back of a list as well
$result = join("+", @fields, "");
# "" is treated as the empty element to be glued
# with the last  data element of @fields
# $result is "Betty Boop+555-5555+1 Boop Lane+100000+";
```