Perl Regular Expressions

- Unlike most programming languages, Perl has built-in support for matching strings using regular expressions called patterns, which are similar to the regular expressions used in Unix utilities, like grep.
- Can be used in conditional expressions and will return a true value if there is a match. Forms for using regular expressions will be presented later.
- Example:

```
if (/hello/) # sees if “hello” appears anywhere in $_
```

Perl Patterns

- A Perl pattern is a combination of:
  - literal characters to be matched directly
  - '.' matches any single character but a newline
  - '*' match the preceding item zero or more times
  - '+' match the preceding item one or more times
  - '?' match the preceding item zero or one times
  - '(' and ')' for grouping
  - '|' match item on the left or item on the right
  - [...] match one character inside the brackets

Examples of Perl Patterns

```
/abc/             # abc
/a.c/             # a, any char but newline, c
/ab?c/            # ac or abc
/ab*c/            # a, zero or more b's, c
/ab|cd/           # ab or cd
/a(b|c)d/         # abd or acd
/a(b|c)+d/        # a, one or more b's or c's, d
/a[bcd]e/         # abe or ace or ade
/a[A-Za-z0-9]b/   # a, letter or digit, b
/a[^A-Za-z]b/     # a, any character but a
                  # letter, b
```

Character Class Shortcuts

- Perl provides shortcuts for commonly used character classes.

```
digit char:       \d == [0-9]
word char:        \w == [A-Za-z0-9]
whitespace char:  \s == [\f\t\n\r ]
nondigit:         \D == [^\d]
nonword:          \W == [^\w]
non whitespace:   \S == [^\s]
```

General Quantifiers

- Can use {min,max} to represent the number of repetitions for an item in a regular expression.

```
a{1,3}     # a, aa, or aaa
a{5,5}     # aaaaa
a{5}       # aaaaa
a{2,}      # two or more a's
a{0,}      # a*
a{1,}      # a+
a{0,1}     # a?
```

Anchors

- Perl anchors provide context in which a pattern is matched.

```
/^a/       # matches a if after beginning of line
/a$/       # matches a if before end of line
/^a$/      # matches a if it is a complete line
/\ba/      # matches a if at the start of a word
/a\b/      # matches a if at the end of a word
/\ba\b/    # matches a if a complete word
```

Remembering Substring Matches

- (...) is used for not only grouping, but also for remembering substrings in a pattern match. Note there are similar features in the sed Unix utility.
- Can refer to these substrings.
  - Backreferences can be used inside the pattern to refer to the memory saved earlier in the current pattern.
  - Memory variables can be used outside of the pattern to refer to the memory saved in the last pattern.

Backreferences

- A backreference has the form *number*. It indicates the string matching the memory reference in the current pattern identified by that *number*. In numbering backreferences, you can just count the left parentheses.

```
/(a|b)\1/     # match aa or bb
/((a|b)c)\1/  # match acac or bcbc
/((a|b)c)\2/  # match aca or bcb
/(.)\1/       # match any character but newline that
              # appears twice in a row
/(\w+)\s+\1/  # match any word that appears twice in a
              # row and is separated by one or more
              # whitespace chars
/(['“]).*\1/  # match string enclosed by '...' or
              # “...”
```

Memory Variables

- A memory variable has the form $*number*. It indicates the string in the last pattern matching the memory reference identified by that *number*.

```
# Checks if $_ has a word and prints that word.
if ( /\s+(\w+)\s+/ ) {
    print $1, “\n”;
}

# If $_ has a '$' followed by 1 to 3 digits and
# optionally followed by groups of a comma with
# 3 digits, then print the price.
if ( /(\$\d{1,3}(,\d{3})*)/ ) {
    print “The price is $1.\n”;
}
```

Binding Operator

- So far we have only seen checks for patterns in $_. We can check for patterns in arbitrary strings using the =~ and !~ match operators.
- General form:

  # check if <pattern> match for <string>
  <string> =~ /<pattern>/

  # check if there is not a <pattern> match for <string>
  <string> !~ /<pattern>/

Example of Using Binding Operators

```
# If the user did not specify to exit,
# then print the line.
if ($line  !~  /\bexit\b/) {
    print $line;
}

# If a blank line, then proceed to the
# next iteration.
if ($line  =~  /^$/) {
    next;
}
```

Automatic Match Variables

- A pattern only has to match a portion of a string to return a true value. There are some automatic match variables that do not require parentheses to be specified within the pattern.

  $` # contains portion of the string before the match

  $& # contains portion of the string that matched

  $' # contains portion of the string after the match

Automatic Match Variable Examples

```
# establish relationship
if ( $line  =~  / is the parent of / ) {
   print “$' is the child of $`\n”;
}

# change the assignment operator
if ( $line  =~  /=/ ) {
   print “$`:=$'”;
}

# find the first word in the line
if ( $line  =~  /\b\w+\b/ ) {
   print “$& is the first word in the line.\n”;
}
```

Using Other Pattern Delimiters

- You can use other delimiters besides slashs for patterns, as we saw with the *qw* shortcut for quoted words in a list. If you do use a different delimiter, then you must precede the first delimiter with an *m*. The *m* is optionable when using slashes. Note some delimiters are paired and others are nonpaired.

m/.../	m{...}	m[...]	m(...)
m!...!	m,...,	m^...^	m#...#

- You should probably use slashes unless your pattern contains slashes, as your Perl code will be easier to read.

Example of Using Other Pattern Delimiters

- Sometimes the pattern matching can be more readable when using a pattern delimiter other than a '/' when the pattern contains a '/'.

```
# Search for the start of a URL.
if ($s =~ /http:\/\//)

# Search for the start of a URL.
if ($s =~ m^http://^)
```

Option Modifiers

- There are a set of letters that you can place after the last delimiter in a pattern to indicate how the pattern is to be interpreted.

Modifier	Description
i	case-insensitive matching
s	. now matches newlines as well
g	find all occurrences

Case-Insensitive Matching

- You can make a case-insensitive pattern match by putting 'i' as an option modifier after the last delimiter.

```
/\b[Uu][Nn][Ii][Xx]\b/  # matches the word
                        # regardless of case
/\bunix\b/i             # same as above
```

Matching Any Character

- The '.' character in a pattern indicates to match any character but a newline. By using the 's' option modifier, the '.' character will also match newlines.

```
# Matching a quoted string that could contain
# newlines.
/“(.|\n)*”/

# A more concise pattern.
/“.*”/s
```

Global Pattern Matching

- You can use the 'g' option modifier to find each match of a pattern in a string. Perl remembers the match position where it left off the last time it matched the string and will find the next match. If the string is a variable and it is modified in any way, then the match position is reset to the beginning of the string.

```
# print each acronym in a string on a
# separate line
while ($s =~ /[A-Z]{2,}/g) {
   print “$&\n”;
}
```

Interpolating Patterns

- The regular expressions allow interpolation just as double quoted strings. Thus, patterns could be read in at run time and used to match strings.

```
# match dynamic pattern if it occurs at the
# beginning of a line
if ($line =~ m/^$var/) {
    print $line;
}
```

- Note that the Perl program may fail if the regular expression comprising the pattern is invalid.