LECTURE 12: REGULAR EXPRESSIONS AND TEXT PROCESSING

Adapted from:
Richard J. Barbalace ,Alex Rolfe, Perl Programming,
sipb-iap-perl@mit.edu

QUIZ 2 ON OCTOBER 16

- Perl & Regular Expressions

OUTLINE

- Regexes (REs) perform textual pattern matching
- Regexes maybe quoted in several ways
- Regexes are their own mini-language
  - Match letters, numbers, other characters
  - Exclude certain characters from matches
  - Match boundaries between types of characters
  - Group subpatterns
  - Match repetitions
- Perl has several regex operators

REGULAR EXPRESSION

- A regular expression is a sequence of characters that defines a search pattern or patterns
  - It uses special characters (often called meta characters)and other characters to match text
  - Meta characters: ^$.|{}[]()*+?\
- Some regex operators and functions in Perl
  - m/PATTERN/: the match operator
  - s/PATTERN/REPLACEMENT/: the substitution
  - =~,!~: binding (also called comparison) operators
- It works with many languages/tools

TEXTUAL PATTERN MATCHING

- Regexes perform textual pattern matching
- Regexes ask: Does a string...
  1...contain the letters "dog" in order?
  2 ...not contain the letter "z"?
  3 ...begin with the letters "Y" or "y"?
  4 ...end with a question mark?
  5 ...contain only letters?
  6 ...contain only digits?

- See textual.pl

TEXTUAL PATTERN MATCHING

```
my $string = "Did the fox jump over the dogs?";
print "1: $string\n" if $string =~/dog/;
print "2: $string\n" if $string !~/z/;
print "3: $string\n" if $string =~/^[Yy]/;
print "4: $string\n" if $string =~/\?$/;
print "5: $string\n" if $string =~/^[a-zA-Z]*$/;
print "6: $string\n" if $string =~/^\d*$/;
```

WAYS TO QUOTE REGEXES

- In Perl, regexes may be quoted in several ways
  - Regex quotes are usually slashes (/regex/)
  - May use other quotes with the match operator m
  - Use another way of quotes when matching slashes (m[])
- See quotes.pl

```
my $string = "Did the fox jump over the dogs?";
print "1: $string\n" if $string =~/dog/;
print "2: $string\n" if $string =~m/dog/;
print "3: $string\n" if $string =~m(dog);
print "4: $string\n" if $string =~m|dog|;
$string = "Did the /fox jump over the dogs?";
print "5: $string\n" if $string =~m[/];
$string = "print \'Did the \"fox\" jump over the dogs?\'\n";
print "6: $string\n" if $string !~m[\\];
$string = "print 'Did the \"fox\" jump over the dogs?'\n";
print " 7:$string\n" if $string !~m[\\];
```

CHARACTER CLASSES (OR CHARACTER SETS)

- Regexes are their own mini-language
- Character classes([])
  - A character class denotes a possible set of characters to match, such as letters, numbers, other characters
  - With a character class, one can tell the regex engine to match only one character out of a set of characters
  - Simply place the characters you want to match between square brackets. Order, repetitions are unimportant. [aabcc] same as [cba]
  - Negate a character class with a carat ([^abc]). It excludes the set of characters from matches
  - ^ outside a character class means "the start of a string”
  - ^ inside a character class means "negate"

PROVIDED CHARACTER CLASSES

- Provided character classes include
  - \d and \D for digits, non-digits respectively
  - \w and \W for word (letters, digits, underscore) and non-word characters
  - \s and \S for white-space (space, tab, new line) and non-white-space characters
  - . for any single character except newline
- See charcls.pl

```
my $string = "Did the fox jump over the dogs?";
print "1: $string\n" if $string =~ m/[bdl]og/; # bog, dog, log
print "2: $string\n" if $string =~m/dog[^s]/; # false, has 's'
print "3: $string\n" if $string =~ m/\s\w\w\wp\s/; # ' jump `
print "4: $string\n" if $string =~ m/dog[^z]/; # matches
```

STRING AND WORD BOUNDARIES

- Match boundaries between types of characters
  - ^ matches the start of the string
  - $ matches the end of the string
  - \b matches a word boundary: boundary between a word character and a non-word character
- See boundaries.pl

```
my $string = "Did the fox jump over the dogs?";
print "1: $string\n" if $string =~ m/^[Yy]/; # no match
print "2: $string\n" if $string =~ m/\?$/; # match
print "3: $string\n" if $string =~ m/the\b/; # match
```

QUANTIFIERS

- Quantify matches
  - * matches the preceding item 0 or more times
  - + matches the preceding item 1 or more times
  - ? matches the preceding item 0 or once(i.e., optional)
  - {4} matches exactly 4 times
  - {3,6} matches 3 to 6 times
- See quantify.pl

GROUPS

- Group subpatterns: () group a subpattern
- Match repetitions
  - 1\ and \2 refer to the first and second groups in the pattern
- See groups.pl

OPERATORS

- Perl has several regex operators
  - m just matches, returning Boolean
  - s/match/replacement/ substitutes
  - tr/class/replacement/ transliterates
- See operator.pl

MODIFIERS

- Perl has several regex modifiers
  - g is global, allows for multiple substitutions
  - i is case insensitive
  - s treats string as one line
- See modifiers.pl

IN PYTHON?

- import re
- re.match(pattern, sequence)
  - Can be casted to bool, or/and used in a conditional
  - Searches only at the beginning
- re.search(pattern, sequence).group()
  - Searches anywhere
- re.findall(pattern, sequence)
  - Finds all matches
- re.compile(pattern)
  - In case RE is reused
  - re_compiled = re.compile(pattern)
    re.search(re_compiled, sequence)
- re.sub(pattern, repl, sequence)

MORE EXAMPLES

Source: Sam Hughes, Learn regular expressions in about 55 minutes
http://qntm.org/files/re/re.html

LITERALS AND THE DOT

- cat: find a c, followed by a, followed by t
- c.t: find a c, followed by any single character except newline, followed by t
- c\.t: c, followed by dot, followed by t
- c\\t c, followed by backslash, followed by t
- c[.]t or c[\.]t same as c\.t

CHARACTER CLASS EXAMPLES

- Order and repetitions are not important
  - [aabbbccc] is the same as [cba]

- c[aeiou]t: c followed by a vowel followed by t (i.e., cat,cet,cit,cot,cut)

- [0123456789]: find a digit, same as \d or [0-9]

- \[a\] find a left square bracket followed by an a followed by a right square bracket

CHARACTER CLASS RANGE AND NEGATION

- [A-Z]: find an upper-case letter
- A-Z: find an A followed by a hyphen followed by a Z
- [0-9.,]: find a digit or a full stop or a comma
- [0-9a-fA-F]: find a hexadecimal digit
- [a-zA-Z0-9\-]: find an alphanumeric character or a hyphen
- [1-31]: find a 1 or 2 or a 3
- [^a]: find any character other than an a
- [^a-zA-Z0-9]: find a non-alphanumeric character

MULTIPLIERS

- a{1}: find an a
- a{3}: find ”aaa”
- [abc]{2}: find a or b or c, followed by a or b or c

FIN!