# Day 7:
# Regular Expressions, Part 1

Suggested Reading:
Learning Perl (4th Ed.),
Chapter 7: In the World of RegExps
Chapter 8: Matching with RegExps

# Reminders

- Turn in homework at **START** of class
- Writing code is **fun**!
  - Write at least a little every day
  - The more you do, the easier it gets
- When in doubt, ask questions!
  - Zathras does not want you being confused
- This is a recording

# What is a Regular Expression?

- Provide a concise and flexible means for identifying strings of text of interest
  - Particular characters
  - Words
  - Patterns of characters
- Written in a formal language that can be interpreted by a regular expression processor, a program that either serves as a parser generator or examines text and identifies parts that match the provided specification
- Abbreviations: regex or regexp
- Plural abbreviations: regexes, regexps, or regexen

Source: `http://en.wikipedia.org/wiki/Regular_expression`, 16 July 2009

# Example Uses

- The sequence of characters "car" in any context, such as "car", "cartoon", or "bicarbonate"
- The word "car" when it appears as an isolated word
- The word "car" when preceded by the word "blue" or "red"
- A dollar sign immediately followed by one or more digits, and then optionally a period and exactly two more digits

Source: `http://en.wikipedia.org/wiki/Regular_expression`, 16 July 2009

# Basic Perl Syntax

- Simple usage:

```
if($line =~ /expression/) {
}
```

- $_ is assumed:

```
if(/expression/) {
}
```

- Common idiom:

```
while(<>) {
  if(/expression/) {
    # do something
  }
}
```

# Basic Perl Examples

- Match:
  - The sequence of characters "car" in any context, such as "car", "cartoon", or "bicarbonate"

```
if(/car/) {
  print;
}
```

# Testing Your Regex

- Use 'perl -ne' to try it out:
  - From the shell:

```
$ perl -ne 'if (/expr/) { print; }'
```

  - For example, the "cat" expression:

```
$ perl -ne 'if (/cat/) { print ":$_"; }'
abc
catalog
:catalog
cat
:cat
dog
```

# Meta Characters

- Characters whcih save special meanings in regular expressions
  - \ Quote the next metacharacter
  - ^ Match the beginning of the line
  - . Match any character (except newline)
  - $ Match the end of the line (or before newline at the end)
  - | Alternation
  - ( ) Grouping
  - [ ] Character class

# Modifiers

- `*` Match 0 or more times
- `+` Match 1 or more times
- `?` Match 1 or 0 times
- `{n}` Match exactly n times
- `{n,}` Match at least n times
- `{n,m}` Match at least n but not more than m times

# Metacharacters: Anchors '^' and '$'

- Regexs can match strings that occur anywhere in the text line:
  - `/abc/`:
    - Match: `"abc"`, `" abc"`, `"blabc"`, `"abcdef"`
- `^` anchors the regex to the start of the line
  - `/^abc/`:
    - Match: `"abc"`, `"abcdef"`
    - Not: `" abc"`, `"blabc"`
- `$` anchors the regex to the end of the line
  - `/abc$/`:
    - Match: `"abc"`, `" abc"`, `"blabc"`
    - Not: `"abcdef"`, `"abc "`

# Meta Character: .

- "." matches any single character
  - /a.c/:
    - Match "abc", "a c", "a.c", "a%c"
    - Not: "abbc", "ac", "a..c"
- With anchors:
  - /^a.c/:
    - Match "abc", "a c", "a.c", "a%c"
    - Not: " abc", "-a.c", "a..c"
  - /a.c$/:
    - Match "abc", "a c", "a.c", "a%c"
    - Not: "abc ", "a.c:", "a..c"

# Meta Character: \

- Used to "escape" the next metacharacter, so it's used like a normal character
  - "\." match only "." chars
  - /a\.c/:
    - Match "a.c", " a.c", "a.c "
    - Not: "abbc", "ac", "a..c", ", "abc", "a c", "a%c"
- Can be used to escape itself:
  - /a\\c/:
    - Match "a\c", "a\c xyz"
    - Not: "abc", "a/c", " abc", "-a.c", "a..c"
- Can be used to escape "/":
  - /a\/c/:
    - Match "a/c", "aaaa/c", " a/c"
    - Not: "abc", "a c", "abc ", "a\c:", "a..c"

# Modifier: *

- *: Match preceding item any number of times
- /.*/:
  - Will match any character, any number of times
  - The "any old junk" pattern
- /a.*c/:
  - Match: "ac", "abc", "a c", "abcabc"
  - Not: "a", "c", "ca"
- /ab*c/:
  - Match: "ac", "abc", "abbbbc", "acc"
  - Not: "a" "c", "a.c", "adc"
- /a\.*c/:
  - Match: "ac", "a.c", "acc", "a...c", "aaac", "aaa...c"
  - Not: "a" "c", "abc", "abc", "abbbbc"

# Modifier: +

- +: Match preceding item one or more of times
- /.+/:
  - Will match any character, one or more times
- /a.+c/:
  - Match: "abc", "abbc", "a c", "abababc", "axxc"
  - Not: "ac", "c", "ca"
- /ab+c/:
  - Match: "abc", "abbbbc"
  - Not: "ac", "acc", "a", "c", "a.c", "abxc", "adc"
- /a\.+c/:
  - Match: "a.c", "a...c"
  - Not: "ac", "abbc", "c", "ca", "a.xc", "ax.c", "a...xc"

# Modifier: ?

- ?: Match preceding item zero or one times
- /.?/:
  - Will match any character, zero or one times
- /a.?c/:
  - Match: "abc", "a c", "axc", "ac"
  - Not: "abbc", "c", "ca", "a bc"
- /ab?c/:
  - Match: "abc", "ac", "acc"
  - Not: "a", "c", "abbc", "adc"
- /a\.?c/:
  - Match: "a.c", "ac"
  - Not: "abbc", "c", "ca", "a.xc", "ax.c", "a...xc"

# Modifier: {}

- {n}: Match preceding item exactly n times
- {n,}: Match preceding item at least n times
- {n,m}: Match preceding item at least n but not more than m times
- /a.{2}c/:
  - Match: "abbc", "ab c", "a bc", "a..c"
  - Not: "abc", "a", "ac", "a c", "axxxc"
- /ab{2,}c/:
  - Match: "abbc", "abbbbc"
  - Not: "abc", "ac", "a bc", "adc"
- /a.{1,2}c/:
  - Match: "a.c", "abc", "abbc", "a:c", "a:bc", "ab c"
  - Not: "ac", "c", "ca", "a...c", "abbbc", "a::bc"

# Options

- Can be used to modify how the expression is evaluated
  - `/<expr>/<opts>`
- i: Case insensitive matching
  - `/abc/i`
    - Match: `"abc"`, `"aBc"`, `"ABC"`
    - Not: `"bac"`, `"CaB"`
- Also: x and s

# Basic Character Classes

- \w Match a "word" character (alphanumeric plus "_")
- \W Match a non-"word" character
- \s Match a whitespace character
- \S Match a non-whitespace character
- \d Match a digit character
- \D Match a non-digit character

# Character classes: \s

- \s matches "whitespace" characters
  - space (" "), tab ("<*tab*>" or "\t"), newline ("<*newline*>" or "\n")
  - /^\s+/ will match any line that starts with whitespace:
    - Match: "  ", " abcdef", "<*tab*>abc"
    - Not: "", ".", "abc ", "*<*newline*>"
    - Note: Equivilent to /^\s/

# Character classes: \s and \S

- \S matches any non-whitespace characters
  - /^\S+/ will match any line that starts with a non-whitespace:
    - Match: "a", ".", "1", "* "
    - Not: " ", "<*tab*>abc"
    - Note: Equivilent to /^\S/

# Character classes: \w and \W

- \w matches "word" characters
  - Alphanumerics + "_"
  - /^\w+/:
    - Match: "a", "abcdef", "ABC ", "abc++", "123", "_abc"
    - Not: "", ".", " abc", "*", "-AbC", "+x"
    - Note: Equivilent to /^\w/
- \W matches non-word characters
  - /^\Wabc$/:
    - Match: " abc", "*abc", ":abc", "+abc"
    - Not: "a", "abcdef", " ABC ", "123", ":abc++", "_abc"

# Character classes: \d and \D

- \d matches "digit" characters
  - 0 1 2 3 4 5 6 7 8 9
  - /^\d+/:
    - Match: "123", "1a", "123ABC "
    - Not: "", ".", " 12abc", "a123", "-5", "+x"
    - Note: Equivilent to /^\d/
- \D matches non-digit characters
  - /^\D+0$/:
    - Match: "c0", "*000", ":abc0", "+abc0"
    - Not: "0a", "abcde0f", " ABC ", "1230", ":abc++", "_abc"

# Custom Character Classes []

- `[<list>]` is used to specify a list of characters (or classes) to match
  - `[c1-c2]` specifies a range of characters
  - `/^[a-z\d]+/`:
    - Match: `"1"`, `"abc2345"`, `"123"`, `"3456abc"`, `"a1"`, `"abc"`
    - Not: `" 1"`, `"A1"`, `".234"`, `"abc_345"`
  - `/^[a-z]+$/i`:
    - Match: `"abc"`, `"ABC"`, `"aBcD"`
    - Not: `"123a"`, `"a1"`, `"1"`, `"abc "`

# Custom Character Classes [^]

- `[^<list>]` is used to specify a list of characters (or classes) to *not* match
  - `/^[^a-z]+/`:
    - Match: `"1"`, `" abc"`, `"&"`, `"123"`, `"3456abc"`, `"Z123"`
    - Not: `"a"`, `"a1"`, `"abc"`, `""`
  - `/^[^\d][\d\.]+/`:
    - Match: `"a123"`, `"a1"`, `".1"`, `"A.1"`, `"%.1"`, `"a1.1"`
    - Not: `"123a"`, `"12"`, `"1"`, `"abc"`, `"abc1"`

# Matching with m//

- With */expr/*, any slashes in the expression need to be escaped
  - `/\/home\/foo\/data\/file01\.txt/`
  - `/http:\/\/www\.cs\.wisc\.edu\/nleroy/`
- More clear: use the `m//` operator
  - Can use chars other than '/':
    - `m|/home/foo/data/fileio\.txt|`
    - `m!/http://www\.cs\.wisc\.edu/~nleroy!`
  - Must escape if you're using that char, though!
    - m|ab\|c|
    - m!ab\!c!

# Class Exercise 1

- `/^\d\d:\d\d[ap]/`
  - "8:1"
  - "12:34"
  - "1:23a"
  - "34:56"
  - "23:45"
  - "02:34p"
  - "29:59a"
  - "56:78"

# Class Exercise 2

- `/^\d{1,2}:\d{2}[ap]?/`
  - "8:1"
  - "12:34"
  - "1:23a"
  - "34:56"
  - "23:45"
  - "02:34p"
  - "29:59a"
  - "56:78"

# Class Exercise 3

- `/^[012]?\d:[0-5]\d[ap]?/`
  - "8:1"
  - "12:34"
  - "1:23a"
  - "34:56"
  - "23:45"
  - "02:34p"
  - "29:59a"
  - "56:78"