Day 9: Regular Expressions

Suggested reading: *Learning Perl* (6th Ed.)

Chapter 7: In the World of Regular Expressions
Chapter 8: Matching with Regular Expressions

Homework Review

Patterns

Can You Identify a Phone Number?

```
Tim's office
24002
608-262-4002
(608) 262-4002
608/262 4002
6 \0/ 8-2-6-2-4 \0/ (02)
+1 (608) 262 4002
6082624002
6,082,624,002
000-000-0000
193-241-8827
```

Some Other (Possible) Patterns

- Telephone numbers (NANP)
- Dates (e.g., 22 July 2011, 2011-07-22)
- Image filenames (e.g., cs-logo.png)
- Hostnames
- Email addresses (***VERY*** hard)
- Specific data records
- Specific lines from a log file

Regular Expressions

A **regular expression** is
a **formal** description
of a **pattern**
that **partitions** all **strings**
into **matching** / **non-matching**

Matching Patterns

```
#!/usr/bin/perl
use strict;
use warnings;

print 'Enter reg. expression (no delimiters): ';
chomp(my $re_string = <STDIN>);
my $re = qr/$re_string/;

open(INPUT, '<', $ARGV[0])
    or die "Could not open file: $!\n";
while (<INPUT>) {
    print if /$re/;
}
close INPUT;
```

Matching Basics

Metacharacters I

Most characters match self (letters, digits, `!`, `@`, …)

`/cat/`	`cat, a cat, catalog, scatter, tomcat` *empty string*, `a, at, act, cart, Cat`

`^` matches start of line

`/^cat/`	`cat, catalog, cathedral, cat's meow`
	`^cat, a cat, scatter, tomcat, ␣cat`

`$` matches end of line

`/cat$/`	`cat, bobcat, scat, tomcat, nice cat`
	`cat$, cats, scatter, cat␣`

`/^cat$/`	`cat` *does not match anything else*

Metacharacters II

`.` matches any *single* character

`/d.g/`	`dog, dig, d.g, adage, mid-game, add2go`
	`Dog, drag, edge, add-2-go`

`\` makes following metacharacter “normal”

`/1\.0/`	`1.0, 131.0.73.12, $21.03`
	`1\.0, 120, 1e0, 10.1`

`/2\^8/`	`2^8`
	`2\^8, 2\8`

`/C:\\/`	`C:\Documents, file:///C:\Documents, C:\\`
	`c:\..., C:foo`

Counting Modifiers I

***** match 0–*n* times (aka “maybe some …”)

```
/an*y/      any, canyon, botany, granny, days, play
            an*y, a, n, y, an, andy, an-y
```

+ match 1–*n* times (aka “some …”)

```
/an+y/      any, canyon, botany, granny, tannyl
            an+y, days, play, Any, a+y
```

? match 0–1 times (aka “maybe a …”)

```
/an?y/      any, canyon, botany, days, play
            an?y, a, n, y, an, andy, ann, granny
```

Counting Modifiers II

`.*` *and* `.+` give you superpowers

`/a.*z/`	`azimuth, dazzle, waltz, abuzz, a.*z`
	`a, z, apples, buzz, Azimuth`

`/a.+z/`	`dazzle, waltz, abuzz, a.*z`
	`a, z, azimuth, apples, buzz, Abuzz`

`{n,m}` match *n*–*m* times; also: `{n}` `{n,}` `{,m}`

`/^a.{3,6}e$/`	`above, ashore, achieve, airframe`
	`ae, ate, able, manager`

Character Classes I

[…] matches *one of* enclosed chars (use **-** for range)

`/q[aeio]/`	`Iraqi, qanat, qintar`
	`q[aeio], q, queue, question, q?`

`/:[0-5][0-9]/`	`1:00, 11:50 a.m., 12:59, page:08`
	`1:60, 2:3 ratio, 256, 42, :`

[^…] matches one of *anything but* enclosed chars

`/q[^u]/`	`Iraqi, qanat, qintar, miqra, q[^u]`
	`q, queue, question`

`/^[^A-Za-z]+$/`	`1, 1:23, 1,234,567, :), \@/, ^_^`
	`^[^A-Za-z]+$, word, 11:50 a.m.`

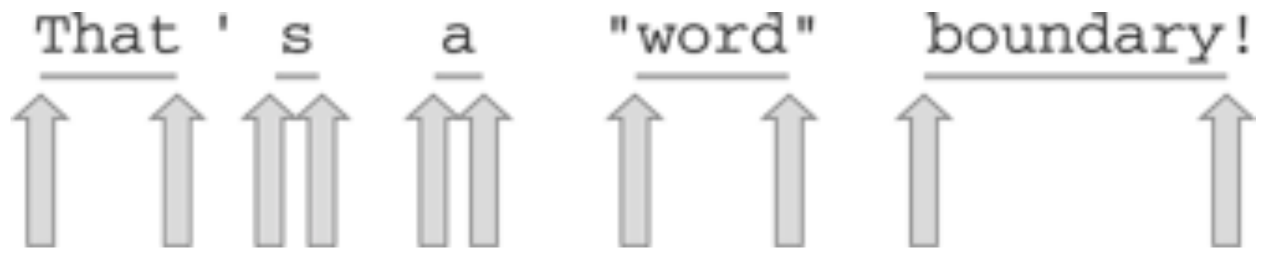
Character Classes II

`\d`	matches a digit (= `[0-9]`)
`\D`	matches a non-digit (= `[^0-9]` or `[^\d]`)
`\w`	matches a "word" char (= `[A-Za-z0-9_]`)
`\W`	matches a non-"word" char (= `[^\w]`)
`\s`	matches whitespace (=`[ \t\n…]`)
`\S`	matches non-whitespace (= `[^\s]`)

`/^-?\d+$/`	`0, 1, -1, 1234, -000` `--1, a1, 1e4, 1.0,` *empty string*
`/^\s*word/`	`word`, maybe with some whitespace before `this line has a word`

Boundaries

`\b`	matches a word *boundary*
`\B`	matches a non-word boundary



`/word\b/`	`word, reword, sword`
	`wordy, wordless, swordplay`

`/\bword\B/`	`wordy, wordless, wordplay`
	`word, sword, swordplay`

Case-Insensitivity

`/…/i` ignore case in matching

`/cat/`	`cat, a cat, catalog, scatter, tomcat`
	`Cat, a Cat, Cathy, TomCat`

`/cat/i`	`cat, Cat, Cathy, tomcat, TomCat`
	`dog`

Commenting Regular Expressions

`//x` Whitespace and comments allowed in RE
Both must be quoted with \ to be part of RE

```
$text =~ s{
    (                      # start of opening
      <hostname>           # open hostname element
      \s *                 # maybe some whitespace
    )                      # end of opening
    . * ?                  # capture hostname here
    (                      # start of closing
      \s *                 # maybe some whitespace
      </hostname>          # end hostname element
    )                      # end of closing
}
{$1$host$2}imx;
```

Delimiters

```
print if  /cat/i;  # checks $_ for match
print if m/cat/i;
print if m,cat,i;
print if m{cat}i;

print if $some_string =~  /cat/i;
print if $some_string =~ m/cat/i;
print if $some_string =~ m,cat,i;
print if $some_string =~ m{cat}i;
```

Other Scripting Languages

- Most have regular expressions
- Perl has the best, by far (cf. PCRE library)
- Others may have limited REs or different syntax
- OO languages often have match objects

Homework

- ***No Perl coding*** — just use provided script
- Write regular expressions
- Need to get 11 correct expressions for full credit
- Some require that you explain what will and will not match: Provide examples!!!