Day 7: Regular Expressions

Suggested reading: *Learning Perl* (4th Ed.)

Chapter 7: In the World of Regular Expressions
Chapter 8: Matching with Regular Expressions
Chapter 9: Processing Text with Regular Expressions

(sorry)

Turn In Homework

Homework Review

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

Patterns

- Dates (e.g., 22 July 2010, 2010-07-22)
- Telephone numbers (NANP)
- Image filenames (e.g., cs-logo.png)
- Hostnames
- Email addresses (***VERY*** hard)
- Certain data records
- Certain lines from a log file

Typical Usage

Test for match

```
if ($input =~ /q(uit)?/i) {
  exit 0;
}
```

Extract part of match

```
if ($number =~ /(\d{3})-\d{3}-\d{4}/) {
  $area_code = $1;
}
```

Substitute match

```
$username =~ s/[^\w-]+/_/g;
```

Matching Basics

Metacharacters I

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

`/cat/`	**cat**, a **cat**, **cat**alog, s**cat**ter, tom**cat** *empty string*, a, at, act, cart, Cat

^ matches start of line

`/^cat/`	**cat**, **cat**alog, **cat**hedral, **cat**'s meow ^cat, a cat, scatter, tomcat, ␣cat

$ matches end of line

`/cat$/`	**cat**, bob**cat**, s**cat**, tom**cat**, nice **cat** cat$, 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+$/`	`0, 1, -1, 1234, -000` `--1, a1, 1e4, 1.0,` *empty string*
`/^\s*#/`	comment (only) lines in a Perl script `$x = 1; # comment`

Using Regular Expressions

Testing a Match

```
while (<STDIN>) {                # each line in $_
  print if /q[^u]/;
}
```

=~ makes a regular expression apply to a variable

```
if ($address =~ /^\d+ \w+/) { … }
```

changing delimiters

```
if ($url =~ m/^https?:\/\/.+$/) { … }
if ($url =~ m,^https?://.+$,) { … }
```

Splitting on a Match

the inverse of **`join()`**:

```
my @items = split /\s*,\s*/, $string;
```

```
'cat'              => ( 'cat' )

'cat, dog'         => ( 'cat', 'dog' )

'1,2 ,3  ,  4'     => ( '1', '2', '3', '4' )
```

Substituting a Match I

`s/…/…/` substitutes a match with a string… ***once***

```
$path =~ s|^.*/|/home/cat/|;
/usr/share/dict/words => /home/cat/words
```

`s/…/…/g` substitutes throughout the string

```
$username =~ s/\W+/_/g;
c@$t!@#$%x => c_t_x
```

Substituting a Match II

pattern interpolation

```
$dir_pattern = '.*/';
$path =~ s|^$dir_pattern|/home/cat/|;
```

substitution interpolation

```
$home_dir = '/home/cat';
$path =~ s|^.*/|$home_dir/|;
```

empty substitutions and ignoring case

```
$no_vowels =~ s/[aeiou]//ig;
```

Groups & Captures

Groups

(…) groups a match

```
/(in){2}/   dining, feminine
            in, ini, nine
```

matching groups are remembered in **$1**, **$2**, …

```
if ($phone =~ /(\d{3})-\d{3}-\d{4}/) {
  $area_code = $1;
}
```

can use captures in substitutions

```
$string =~ s/(\w+), (\w+)/$2, $1/;

'a big, great cat' => 'a great, big cat'
```

Alternatives

(…|…) matches one or the other

```
/d(og|im)/   dog, dim, dime
             dom, dig
```

( ) optional when | applies to whole pattern

```
/here|hear/   here, there, hear, heart, gatherer
              her, har, ...
```

gets messy…

```
if (/^((1[0-2])|([1-9])):([0-5]\d)$/) {
  print "hour: $1, minute: $4\n";
}
```

Now *YOU* Can Do This:

http://xkcd.com/208/

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

- Analyze a Perl script
  - Count "real" lines of code
  - Find all defined subroutine names
  - Find all scalar variable names
- Use regular expressions!
- **Get as close as you can, don't worry if not perfect**
- Read the homework assignment for hints