# Day 9: Regular Expressions II

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

### Chapter 8: Matching with Regular Expressions

### Chapter 9: Processing Text with Regular Expressions

# Turn In Homework

# Homework Review

# Matching

# Thus Far…

| | |
|---|---|
| *letters*, `!`, `@`, … | match self |
| `^` | start of string |
| `$` | end of string |
| `.` | any character |
| `\`***x*** | *x*, without special meaning |
| `*` | match preceding, 0–*n* |
| `+` | match preceding, 1–*n* |
| `?` | match preceding, 0–1 |
| `{`***n***`,`***m***`}` | match preceding, *n*–*m* |
| `[…]` & `[^…]` | character classes |
| `\d`, `\D`, `\w`, `\W`, `\s`, `\S` | digit, word char, whitespace |

# Greedy vs. Non-Greedy Matches

## .* greedy match

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

## .*? non-greedy match

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

- Append **?** to *, **+**, **?**, **{}**
- Good for delimited text

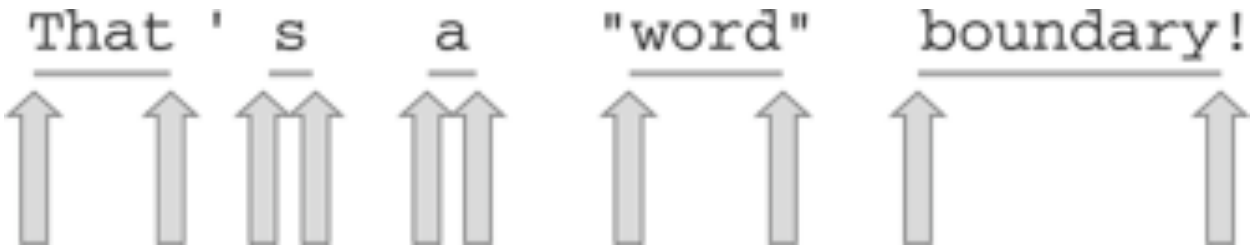
```
<img src="/icon.png" alt="icon">

/<img\s+src="(.+?)"/
```

# Boundaries

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

word boundary: *between* **\w** and **\W** (or vice versa)




```
/word\b/
```

```
word, reword, sword, word's
wordy, wordless, swordplay
```

```
/\bword\B/
```

```
wordy, wordless, wordplay
word, sword, swordplay
```

# Groups

**(…)** groups a match

```
/(in){2}/
```

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

```
/^(un)?ar/
```

artful, unary, arm, unarmed
narrow, lunar, uar, (un)?ar

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

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

```
/^(.)(.)\2\1$/
```

anna, deed, maam, noon
*not much else…*

# Alternatives

**(aaa|bbb)** matches all of "aaa" ***or*** all of "bbb"

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

**( )** optional when **|** applies to whole pattern

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

Groups and alternatives are powerful (if complex)

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

# Modifiers

# Ignoring Letter Case

`//i` Ignore case in matching

| | |
|---|---|
| `/cat/` | `cat, a cat, catalog, scatter, tomcat`<br>`Cat, a Cat, Cathy, TomCat` |
| `/cat/i` | `cat, Cat, Cathy, tomcat, TomCat`<br>`dog` |

# Commenting Regular Expressions

//x Whitespace and comments allowed in RE
Quote both with \ to include in RE itself

```
print if /
  ^                 # start of string
  (                 # group hour part ($1)
    (1[0-2])        # hours option 1: 10-12 ($2)
    |
    ([1-9])         # hours option 2: 1-9 ($3)
  )                 # end of hour group
  :
  ([0-5]\d)         # minutes: 00-59 ($4)
  $                 # end of string
/x;
```

# Matching Across Lines

`//m` Treat string as multiple lines

`^`, `$` match start, end of any line within string

`//s` Treat string as single line

`.` matches newline (which normally it does not)

```
<person><name>
  Tim
  Cartwright
  </name><office>
4265</office></person>

/<name>\s*(.*?)\s*<\/name>/im
```

# Back to Perl

# Matching

```
print if  /cat/i;
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;
```

# Extracting

```
print "Enter your birthday (YYYY-MM-DD): ";
my $date = <STDIN>;
my ($year, $month, $day);
if ($date =~ /(\d{4})-(\d{2})-(\d{2})/) {
    $year  = $1;
    $month = $2;
    $day   = $3;
}
```

```
if ($xml =~ m{<name>\s*(.*?)\s*</name>}im) {
    $name = $1;
} else {
    print "Could not extract name from XML.\n";
}
```

# Splitting

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

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

```
$text = 'Of France and England, ...';
$text =~ s/and/or/;

=> 'Of France or England, ...'
```

**s/…/…/g** substitutes all matches in the string

```
$text = 'Of France and England, ...';
$text =~ s/and/or/g;

=> 'Of France or Englor, ...'
```

Can use groups in substitution

```
$html =~ s,<i>(.*?)</i>,<em>$1</em>,g;
```

# Substitution Examples

Remove all vowel characters, regardless of case:

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

Make every line of code a comment, if not already:

```
while (<INPUT>) {
    s/^(\s*)([^#\s])/$1# $2/;
    print;
}
```

Do something if any substitutions occurred:

```
if ($text =~ s/Prof\w* Cartwright/Tim/g) {
    print "Incorrect title purged!\n";
}
```

# Variable Interpolation

Perl interpolates variables into a regular expression ***before*** processing the expression itself

Interpolation in the pattern:

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

Interpolation in the substitution

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

# Now *YOU* Can Do This:


http://xkcd.com/208/

# More Resources for Regular Expressions

- [Madcat] ***Mastering Regular Expressions***, Friedl
- [Madcat] ***Perl Cookbook***
- Google for patterns
  - Can be very helpful
  - Do you trust what you find?
  - Understand assumptions, limitation, etc.
  - Use as inspiration, not as copy-and-paste solution

# Homework

- Analyze ***and*** modify a Perl script, programmatically
- Reports
  - Many to choose from
  - Need only ***two*** for full credit
- Modifications
  - A few to choose from
  - Need only ***one*** for full credit
  - Write modified script out to new file