

# Day 7: Regular Expressions

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

Chapter 7: In the World of Regular Expressions

Chapter 8: Matching with Regular Expressions

# Turn In Homework

# 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;

open(INPUT, '<', $ARGV[0])
  or die "Could not open file: $!\n";

while (<INPUT>) {
  print if /cat/;
}

close(INPUT);
```

# Matching Basics

## Metacharacters I

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

**/cat/**      **cat**, a **cat**, **catalog**, **scatter**, tom**cat**  
*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**, bob**cat**, **scat**, tom**cat**, nice **cat**  
**cat\$**, cats, scatter, cat**\_**

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

## Metacharacters II

- matches any *single* character

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

- `\` makes following metacharacter “normal”

<code>/1\.0/</code>	<code>1.0, 131.0.73.12, \$21.03</code> <code>1\.0, 120, 1e0, 10.1</code>
---------------------	---

<code>/2\^8/</code>	<code>2^8</code> <code>2\^8, 2\8</code>
---------------------	--

<code>/C:\\</code>	<code>C:\Documents, file:///C:\Documents, C:\\</code> <code>c:\..., C:foo</code>
--------------------	---

## 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)

<code>/q[aeio]/</code>	Iraq <b>i</b> , qan <b>a</b> t, q <b>i</b> ntar q[aeio], q, queue, question, q?
------------------------	--

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

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

<code>/q[^u]/</code>	Iraq <b>i</b> , qan <b>a</b> t, q <b>i</b> ntar, mi <b>q</b> ra, q[^u] q, queue, question
----------------------	--

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

## Character Classes II

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

```
/^-?\d+$/
```

**0, 1, -1, 1234, -000**

**--1, a1, 1e4, 1.0, empty string**

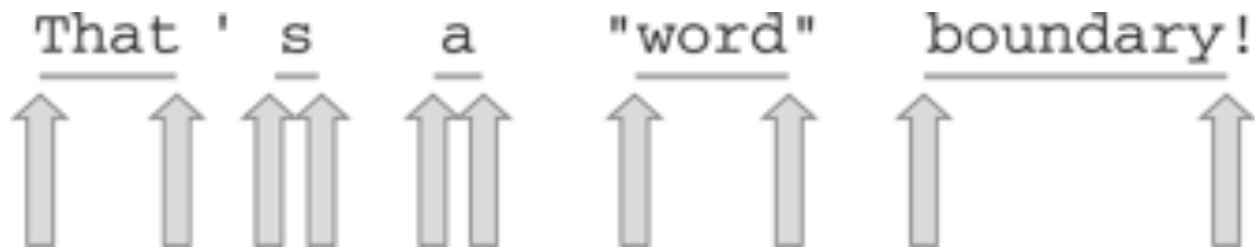
```
/^\s*word/
```

**word**, maybe with some whitespace before  
**this line has a word**



## Boundaries

<code>\b</code>	matches a <i>word boundary</i>
<code>\B</code>	matches a <i>non-word boundary</i>



`/word\b/`

**word**, **reword**, **sword**  
wordy, wordless, swordplay

`/\bword\B/`

**wordy**, **word**less, **word**play  
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;  
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 7 of 10 right for full credit
- Some require that you explain what will and will not match: Provide examples!!!