Code Examples from Lectures on perl_regex

count_enrollments.pl

count how many people enrolled in each course

```
open my $f, '<', "course_codes" or die "$0: can not open course_codes: $!";
while ($line = <$f>) {
    chomp $line;
    $line =~ /([^ ]+) (.+)/ or die "$0: bad line format '$line'";
    $course_names{$1} = $2;
}
close $f;

while ($course = <>) {
    chomp $course;
    $course =~ s/\|.*//;
    $count{$course}++;
}

foreach $course (sort keys %count) {
    print "$course_names{$course} has $count{$course} students enrolled\n";
}
```

count_first_names.pl

run as count_first_names.pl enrollments count how many people enrolled have each first name

```
while ($line = <>) {
    @fields = split /\|/, $line;
    $student_number = $fields[1];
    next if $already_counted{$student_number};
    $already_counted{$student_number} = 1;
    $full_name = $fields[2];
    $full_name =~ /.*,\s+(\S+)/ or next;
    $first_name = $1;
    $fn{$first_name}++;
}

foreach $first_name (sort keys %fn) {
    printf "There are %2d people with the first name $first_name\n", $fn{$first_name};
}
```

duplicate_first_names.pl

run as duplicate_first_names.pl enrollments
Report cases where there are multiple people of the same same first name enrolled in a course

```
while ($line = <>) {
    @fields = split /\|/, $line;
    $course = $fields[0];
    $full_name = $fields[2];
    $full_name =~ /.*,\s+(\S+)/ or next;
    $first_name = $1;
    $cfn{$course}{$first_name}++;
}

foreach $course (sort keys %cfn) {
    foreach $first_name (sort keys %{$cfn{$course}}) {
        next if $cfn{$course}{$first_name} < 2;
        printf "In $course there are %d people with the first name $first_name\n", $cfn{$course}{$first_name};
    }
}
```

gender_reversal.0.pl

For each file given as argument replace occurrences of Hermione allowing for some misspellings with Harry and vice-versa.
Relies on Zaphod not occurring in the text.
Modified text is stored in a new file which is then renamed to replace the old file

```
foreach $filename (@ARGV) {
    $tmp_filename = "$filename.new";
    die "$0: $tmp_filename already exists" if -e "$tmp_filename";
    open my $f, '<', $filename or die "$0: Can not open $filename: $!";
    open my $g, '>', $tmp_filename or die "$0: Can not open $tmp_filename : $!";
    while ($line = <$f>) {
        $line =~ s/Herm[io]+ne/Zaphod/g;
        $line =~ s/Harry/Hermione/g;
        $line =~ s/Zaphod/Harry/g;
        print $g $line;
    }
    close $f;
    close $g;
    rename "$tmp_filename", $filename or die "$0: Can not rename file";
}
```

gender_reversal.1.pl

For each file given as argument replace occurrences of Hermione allowing for some misspellings with Harry and vice-versa.
Relies on Zaphod not occurring in the text.
Modified text is stored in an array then the file is over-written

```
foreach $filename (@ARGV) {
    open my $f, '<', $filename or die "$0: Can not open $filename: $!";
    $line_count = 0;
    while ($line = <$f>) {
        $line =~ s/Herm[io]+ne/Zaphod/g;
        $line =~ s/Harry/Hermione/g;
        $line =~ s/Zaphod/Harry/g;
        $new_lines[$line_count++] = $line;
    }
    close $f;
    open my $g, '>', ">$filename" or die "$0: Can not open $filename : $!";
    print $g @new_lines;
    close $g;
}
```

gender_reversal.2.pl

For each file given as argument replace occurrences of Hermione allowing for some misspellings with Harry and vice-versa.
Relies on Zaphod not occurring in the text.
Modified text is stored in an array then the file is over-written

```
foreach $filename (@ARGV) {
    open my $f, '<', $filename or die "$0: Can not open $filename: $!";
    @lines = <$f>;
    close $f;

    # note loop variable $line is aliased to array elements
    # changes to it change the corresponding array element
    foreach $line (@lines) {
        $line =~ s/Herm[io]+ne/Zaphod/g;
        $line =~ s/Harry/Hermione/g;
        $line =~ s/Zaphod/Harry/g;
    }

    open my $g, '>', ">$filename" or die "$0: Can not open $filename : $!";
    print $g @lines;
    close $g;
}
```

gender_reversal.3.pl

For each file given as argument replace occurrences of Hermione allowing for some misspellings with Harry and vice-versa.
Relies on Zaphod not occurring in the text. text is read into a string, the string is changed, then the file is over-written

See http://www.perlmonks.org/?node_id=1952 for alternative way to read a file into a string

```
foreach $filename (@ARGV) {
    open my $f, '<', $filename or die "$0: Can not open $filename: $!";
    while ($line = <$f>) {
        $novel .= $line;
    }
    close $f;

    $novel =~ s/Herm[io]+ne/Zaphod/g;
    $novel =~ s/Harry/Hermione/g;
    $novel =~ s/Zaphod/Harry/g;

    open my $g, '>', ">$filename" or die "$0: Can not open $filename : $!";
    print $g $novel;
    close $g;
}
```

gender_reversal.4.pl

For each file given as argument replace occurrences of Hermione allowing for some misspellings with Harry and vice-versa.
Relies on Zaphod not occurring in the text.
The unix filter-like behaviour of <> is used to read files
Perl's -i option is used to replace file with output from script

```
while ($line = <>) {
    chomp $line;
    $line =~ s/Herm[io]+ne/Zaphod/g;
    $line =~ s/Harry/Hermione/g;
    $line =~ s/Zaphod/Harry/g;
    print $line;
}
```

gender_reversal.5.pl

For each file given as argument replace occurrences of Hermione allowing for some misspellings with Harry and vice-versa.
Relies on Zaphod not occurring in the text.
The unix filter-like behaviour of <> is used to read files
Perl's -i option is used to replace file with output from the script.
Perl's default variable $_ is used

```
while (<>) {
    s/Herm[io]+ne/Zaphod/g;
    s/Harry/Hermione/g;
    s/Zaphod/Harry/g;
}
```

gender_reversal.6.pl

For each file given as argument replace occurrences of Hermione allowing for some misspellings with Harry and vice-versa.
Relies on Zaphod not occurring in the text.
Perl's -p option is used to produce unix filter-like behaviour.
Perl's -i option is used to replace file with output from the script.

```
s/Herm[io]+ne/Zaphod/g;
s/Harry/Hermione/g;
s/Zaphod/Harry/g;
```

wget.0.pl

Fetch a web page removing HTML tags and constants (e.g &amp;)
Lines between script or style tags are skipped.
Non-blank lines are printed

There are better ways to fetch web pages (e.g. HTTP::Request::Common)
The regex code below doesn't handle a number of cases. It is often better to use a library to properly parse HTML before processing it.
But beware illegal HTML is common & often causes problems for parsers.

```
foreach $url (@ARGV) {
    open my $f, '-|', "wget -q -O- '$url'" or die;
    while ($line = <$f>) {
        if ($line =~ /^\s*<(script|style)/i) {
            while ($line = <$f>) {
                last if $line =~ /^\s*<\/(script|style)/i;
            }
        } else {
            $line =~ s/&\w+;/ /g;
            $line =~ s/<[^>]*>//g;
            print $line if $line =~ /\S/;
        }
    }
    close $f;
}
```

wget.1.pl

Fetch a web page removing HTML tags and constants
The contents of script or style tags are removed..
Non-blank lines are printed

The regex code below doesn't handle a number of cases. It is often better to use a library to properly parse HTML before processing it.
But beware illegal HTML is common & often causes problems for parsers.
note the use of the s modifier to allow . to match a newline

```
use LWP::Simple;
foreach $url (@ARGV) {
    $html = get $url;
    $html =~ s/<script.*?<\/script>//isg;  # remove script tags including contents
    $html =~ s/<style.*?<\/style>//isg;    # remove style tags including contents
    $html =~ s/<.*?>//isg; # remove tags
    $html =~ s/\n\s*\n/\n/ig;  # blank lines
    print $html;
}
```

find_numbers.0.pl

Find the positive integers among input text print their sum and mean

Note regexp to split on non-digits
Note check to handle empty string from split

```
@input_text_array = <>;
$input_text_array = join "", @input_text_array;

@numbers = split(/\D+/, $input_text_array);
print join(",", @numbers), "\n";

foreach $number (@numbers) {
    if ($number ne '') {
        $total += $number;
        $n++;
    }
}

if (@numbers) {
    printf "$n numbers: total $total mean %s\n", $total/$n;
}
```

find_numbers.1.pl

Find integers (positive and negative) among input text print their sum and mean

Note regexp to match number: -?\d+
Harder to use split here (unlike just positive integers)

```
@input_text_array = <>;
$input_text_array = join "", @input_text_array;

@numbers = $input_text_array =~ /-?\d+/g;

foreach $number (@numbers) {
    $total += $number;
}

if (@numbers) {
    $n = @numbers;
    printf "$n numbers: total $total mean %s\n", $total/$n;
}
```

print_last_number.pl

Print the last number (real or integer) on every line if there is one.
Note regexp to match number: -?\d+(\.\d+)?

```
while ($line = <>) {
    if ($line =~ /(-?\d+(\.\d+)?)\D*$/) {
        print "$1\n";
    }
}
```

course_first_names.pl

run as course_first_names.pl enrollments report cases where there are multiple people same first name enrolled in acourse

```
while ($line = <>) {
    @fields = split /\|/, $line;
    $course = $fields[0];
    $full_name = $fields[2];
    $full_name =~ /.*,\s+(\S+)/ or next;
    $first_name = $1;
    $cfn{$course}{$first_name}++;
}

foreach $course (sort keys %cfn) {
    foreach $first_name (sort keys %{$cfn{$course}}) {
        next if $cfn{$course}{$first_name} < 2;
        printf "In $course there are %d people with the first name $first_name\n", $cfn{$course}{$first_name};
    }
}
```

course_statistics.pl

for each courses specified as arguments print a summary of the other courses taken by students in this course

```perl
$enrollment_file = shift @ARGV or die;
$debug = 0;

open my $c, '<', "course_codes" or die "$0: can not open course_codes: $!";
while (<$c>) {
    ($code, $name) = /\s*(\S+)\s+(.*)/ or die "$0: invalid course codes line: $_";
    $course_name{$code} = $name;
    print STDERR "code='$code' -> name='$name'\n" if $debug;
}
close $c;

open my $f, "<$enrollment_file" or die "$0: can not open $enrollment_file: $!";;
while (<$f>) {
    ($course,$upi,$name) = split /\s*\|\s*/;
    push @{$course{$upi}}, $course;
    $name =~ s/(.*), (.*)/$2 $1/;
    $name =~ s/ .* / /;
    $name{$upi} = $name;
}
close $f;

foreach $course (@ARGV) {
    %n_taking = ();
    $n_students = 0;
    foreach $upi (keys %course) {
        @courses = @{$course{$upi}};
        next if !grep(/$course/, @courses);
        foreach $c (@courses) {
            $n_taking{$c}++;
        }
        $n_students++;
    }
    foreach $c (sort {$n_taking{$a} <=> $n_taking{$b}} keys %n_taking) {
        printf "%5.1f%% of %s students take %s %s\n",
            100*$n_taking{$c}/$n_students, $course, $c, $course_name{$c};
    }
}
```