1.

2. What is Unicode

   A standard for representing all the world's written languages and other symbols - it has 100,000+ characters,
   ASCII represents 128 characters including the English alphabet.

3. What is UTF-8

   An encoding for Unicode which uses 1-4 bytes to represnt characters.
   It is backward compatible with ASCII - the 128 ASCII characters have the same value in UTF-8

4. Do I need to I know about UTF-8 for the assignment.

   Yes and no.
   The test data set are encoded in UTF-8 but only a small fraction of the posts/replies/comments contain non-ASCII so you can do a lot of the assignment Asuuming the data is ASCII.

   Most the world's web pages use UTF-8 and for example users of a social media website with non-ASCII names would be (rghtly) upset if their name are shown correctly!

   So if you get past teh first couple of subet you may have to do a little research on handling unicode.

5. Write a Perl program script `distance_from_unsw.pl` which lists the latitude, longitude and suburb (if provided) of all matelook users in a dataset it is given as argument.
   The users must be listed in increasing order of distance from UNSW.

   For example:

```
$ ./distance_from_unsw.pl dataset-small
z3413158 lives at -33.9229 151.2303 in University Of New South Wales
z3466413 lives at -33.9217 151.2247 in University Of New South Wales
z5076002 lives at -33.9103 151.2323 in University Of New South Wales
z5040176 lives at -33.9111 151.2342 in University Of New South Wales
z5059413 lives at -33.9049 151.2433 in Randwick
z5014861 lives at -33.9489 151.2105 in Banksmeadow
z5099187 lives at -33.9838 151.2374 in Little Bay
z3493921 lives at -33.8062 151.2003 in Willoughby
z5063045 lives at -33.9688 151.0567 in Peakhurst
z3462191 lives at -33.7393 150.9988 in Castle Hill
```

   Assume UNSW's (latitiude, longitude) is (-33.9172238,151.2302268).

   Don't print users who don't provide their latitude/longitude.

   You can assume if a user provides either latitude or longitude they provide both.

   Sample solution for distance_from_unsw.pl

```
#!/usr/bin/perl -w

$dataset_dir = $ARGV[0] or die;

$unsw_latitude = -33.9172238;
$unsw_longitude = 151.2302268;

foreach $user_directory (glob "$dataset_dir/*") {
        open F, "<$user_directory/user.txt" or next;
        my $username = $user_directory;
        $username =~ s/.*\///;
        my ($line, $latitude, $longitude);
        while ($line = <F>) {
                if ($line =~ /^home_latitude=(\S+)/) {
                        $latitude{$username} = $1;
                } elsif ($line =~ /^home_longitude=(\S+)/) {
                        $longitude{$username} = $1;
                } elsif ($line =~ /^home_suburb=(.*)/) {
                        $suburb{$username} = $1;
                }
        }
        close F;
}

sub distance_from_unsw {
        my ($username) = @_;
        my ($latitude, $longitude) = ($latitude{$username}, $longitude{$username});
        return sqrt(($latitude - $unsw_latitude) ** 2 + ($longitude - $unsw_longitude) ** 2)
}

@usernames = keys %latitude;
@sorted_usernames = sort {distance_from_unsw($a) <=> distance_from_unsw($b)} @usernames;

foreach $username (@sorted_usernames) {
        printf "$username lives at $latitude{$username} $longitude{$username}";
        print " in $suburb{$username}" if $suburb{$username};
        print "\n";
}
```

6. Translate distance_from_unsw.pl to Python.

Sample solution for distance_from_unsw.py

```
#!/usr/bin/python

import glob, math, os, re, sys
dataset_dir = sys.argv[1]

unsw_latitude = -33.9172238
unsw_longitude = 151.2302268

latitude = {}
longitude = {}
suburb = {}
for user_directory in glob.glob(os.path.join(dataset_dir, "*")):
    username = os.path.basename(user_directory)
    with open(os.path.join(user_directory, "user.txt")) as f:
        for line in f:
            m = re.match(r'(.*?)=(.*)', line)
            if not m:
                continue
            if m.group(1) == 'home_latitude':
                latitude[username] = float(m.group(2))
            elif m.group(1) == 'home_longitude':
                longitude[username] = float(m.group(2))
            elif m.group(1) == 'home_suburb':
                suburb[username] = m.group(2)

def distance_from_unsw(username):
    (lat, int) = (latitude[username], longitude[username])
    return math.sqrt((lat - unsw_latitude) ** 2 + (int - unsw_longitude) ** 2)

usernames = list(latitude.keys())
sorted_usernames = sorted(usernames, key=distance_from_unsw)

for username in sorted_usernames:
    sys.stdout.write("%s lives at %s, %s" % (username, latitude[username], longitude[username]))
    if username in suburb:
        sys.stdout.write(" in %s" % suburb[username])
    sys.stdout.write('\n')
```

7. Write a CGI script which allow users to change what is stored in a file.
   Here is an example implementation:

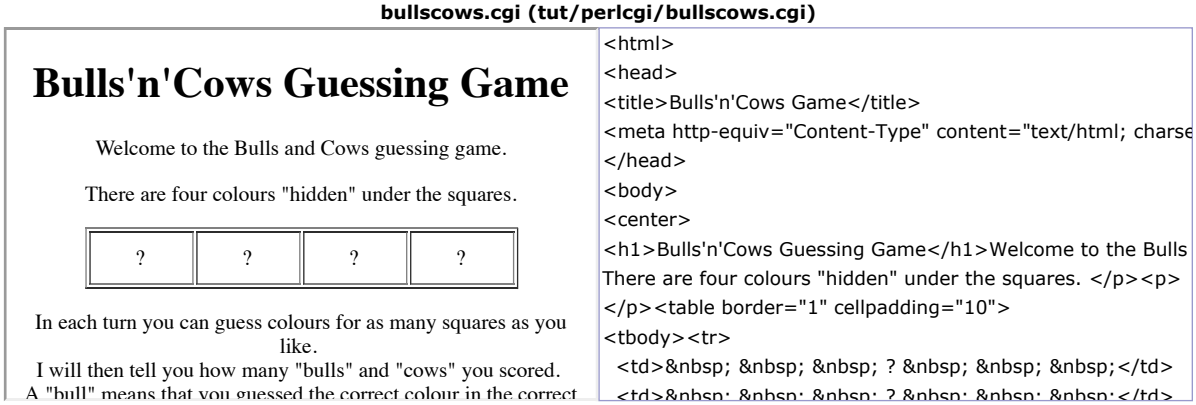
editfile.cgi (tut/perlcgi/editfile.cgi)

Sample solution for editfile.cgi

```
#!/usr/bin/perl -w

use CGI qw/:all/;
use CGI::Carp qw(fatalsToBrowser warningsToBrowser);

# Simple CGI script written by andrewt@cse.unsw.edu.au
# Allow users to change a file

print header, start_html('A Simple Example');
warningsToBrowser(1);

if (param('Save') && defined param('contents')) {
    open FILE, ">example_13.txt" or die "Can not open example_13.txt: $!";
    print FILE param('contents');
    close FILE;
    print h2('Saved'),end_html;
    exit 0;
}

if (!defined param('contents') && open FILE, "<example_13.txt") {
        # Note there is a large risk of security holes if you display user-supplied HTML
        # The substitutions below remove some of the risks
        my $contents = join "", <FILE>;
        $contents =~ s/&/&amp;/g;
        $contents =~ s/</&lt;/g;
        $contents =~ s/>/&gt;/g;
    param('contents', $contents);
}

print   h2('File contents are:'),
        start_form,
        textarea(-name=>'contents', -rows=>10,-cols=>60),
        p, submit('Save'),
        end_form,
        end_html;
```

8. Write a CGI script to play the Bulls and Cows (http://en.wikipedia.org/wiki/Bulls_and_cows) guessing game.
   Here is an example implementation:

bullscows.cgi (tut/perlcgi/bullscows.cgi)



Sample solution

```
#!/usr//bin/perl
#
# Play the Bulls and Cows game
#
# This script has three states:
# - "" (initial state before we start playing the game)
# - Guessing (state for making guess/processing previous guess)
# - Won (state when the player guesses correctly)
#
# States are implemented via the State data item
#
# Other data items that are carried from state to state:
# - Answer (what we're trying to guess)
# - Guesses (comma-separated list of guesses made so far)
# - Box1,Box2,Box3,Box4 (most recent colours guessed for boxes)
#
# All data items are initially null ("")
#

use CGI qw/:all/;

# Information for the current state

$state   = param('State');
$guesses = param('Guesses');
$answer  = param('Answer');

# Constants (colour list and welcome message)

%colours = ("r"=>"Red","y"=>"Yellow","g"=>"Green","b"=>"Blue");

$welcome = <<WELCOME
Welcome to the Bulls and Cows guessing game. <p>
There are four colours "hidden" under the squares. <p>
<table border=1 cellpadding=10>
<tr>
  <td>      ?      </td>
  <td>      ?      </td>
  <td>      ?      </td>
  <td>      ?      </td>
</tr>
</table> <p>
In each turn you can guess colours for as many squares as you like. <br>
I will then tell you how many "bulls" and "cows" you scored. <br>
A "bull" means that you guessed the correct colour in the correct square. <br>
A "cow" means that you guessed the correct colour but in the wrong square. <br>
Each guess will be counted only once, and bulls are counted before cows. <p>
The aim of the game is for you to work out the colours <br>
in the least number of guesses.
WELCOME
;

# Start off each page with a standard header

print(
    header,
    start_html(-title=>'Bulls\'n\'Cows Game'),
    "<center>\n",
    h1("Bulls\'n\'Cows Guessing Game")
);

if ($state eq "")
# Initial state: print welcome/instruction message
{
    # Generate a random sequence of four colours
    @cols = keys %colours;
    foreach $i (1..4) {
        $c = $cols[int(rand($#cols+1))];
        $answer .= $c;
    }

    # Print message, set initial state and display button to start game
    print(
        $welcome,
        start_form,
        "<input type=hidden name='State' value='Guessing'>",
        "<input type=hidden name='Guesses' value=''>",
        "<input type=hidden name='Answer' value='$answer'>",
        submit("Start the game"),
        end_form
    );
}
elsif ($state eq "Guessing")
# Check previous guess (if any)
# If won, then print a message and click to scoreboard page
# If not won, print the guessing table, plus previous guesses
{
    # Process previous guess

    $guess = param('Box1').param('Box2').param('Box3').param('Box4');
    ($bulls, $cows) = &bullcow($guess, $answer);
    $guesses = "$guess,$guesses";
    $guesses =~ s/,$//;

    # We won! So set up for winning state

    if ($bulls == 4)
```

```
    {
        @g = split(/,/,$guesses);
        $ng = $#g+1;
        print(
            "Congratulations! You guessed it.",
            p,
            "It took you $ng guesses.",
            p,
            start_form,
            "<input type=hidden name='State' value='Won'>",
            "<input type=hidden name='Score' value='$ng'>",
            "Enter your name:",
            "<input type=text name='Player'>",
            p,
            submit("Click for ScoreBoard"),
            end_form,
            end_html
        );
        exit 0;
    }

    # Set up table containing pull-down colour menus
    #  for collecting the next guess

    print(
        start_form,
        "<input type=hidden name='State' value='Guessing'>",
        "<input type=hidden name='Guesses' value='$guesses'>",
        "<input type=hidden name='Answer' value='$answer'>",
        "<table border=1 cellpadding=5><tr>"
    );
    foreach $i (1..4) {
        print "<td><select name='Box$i' default='?'>\n";
        print "<option value='?'>?\n";
        foreach $c (keys %colours) {
            print "<option value='$c'>$colours{$c}\n";
        }
        print "</select></td>\n";
    }
    print(
        "</tr></table>\n",
        p,
        submit("Submit guess"),
        end_form,
        p
    );

    # Iterate over previous guesses, displaying
    #  each guess along with the score it obtained

    print h3("Previous Guesses");
    foreach $guess (split(/,/,$guesses)) {
        print "<p><table border=1 cellpadding=5><tr>";
        foreach $c (split(//, $guess)) {
            if ($c eq "?") {
                print "<td>   ?     </td>";
            }
            else {
                print(
                    "<td bgcolor='$colours{$c}'>",
                    "         ",
                    "</td>"
                );
            }
        }
        ($bulls, $cows) = &bullcow($guess, $answer);
        print "<td>$bulls Bulls, $cows Cows</td>";
        print "</tr></table>\n";
    }
}
elsif ($state eq "Won")
{
    # Update the scoreboard

    $score = param('Score');
    $score =~ s/[^\d]//g;             # remove all but expected characters
    $score = substr $score, 0, 4;    # limit score to 4 characters
    $player = param('Player');
    $player =~ s/[^\w\s-_]//g;        # remove all but expected characters
    $player = substr $player, 0, 256; # limit player to 256 characters
    if (!open(SCORES, ">>ScoreBoard")) {
        print h3(font({-color=>'red'},"Can't write Scoreboard"));
    }
    else {
        print SCORES "$score;$player;".localtime()."\n";
        close(SCORES);
    }

    # Fetch the scoreboard information

    if (!open(SCORES, "<ScoreBoard")) {
        print(
            h3(font({-color=>'red'},"Can't read Scoreboard")),
            end_html
        );
        exit 0;
    }
    $i = 0;
```

```
    while ($line = <SCORES>) {
        $lines[$i++] = $line;
    }
    close(SCORES);

    # Display the scoreboard

    print h3("Best Scores");
    print(
        "<table border=1>\n",
        "<tr><th>Score</th><th>Player</th><th>Date</th></tr>\n"
    );
    foreach $line (sort @lines) {
        @bits = split(/;/,$line);
        print "<tr><td>$bits[0]</td><td>$bits[1]</td><td>$bits[2]</td></tr>\n";
    }
    print "</table>\n";

    # Print button for starting a new game

    print(
        start_form,
        "<input type=hidden name='State' value=''>",
        submit("Play another game?"),
        end_form,
    );
}

print "</center>",end_html;

# BullCow subroutine:
#   Takes two strings (each of four chars) and computes the
#   number of direct (Bull) matches and indirect (Cow) matches

sub bullcow()
{
    my @guess  = split(//,$_[0]);
    my @answer = split(//,$_[1]);
    my $bulls = 0;
    my $cows  = 0;

    # Count bulls

    for $i (0..3) {
        if ($guess[$i] eq $answer[$i]) {
            $bulls++;
            $guess[$i] = "?";
            $answer[$i] = "#";
        }
    }

    # Count cows

    foreach $i (0..3) {
        foreach $j (0..3) {
            if ($guess[$i] eq $answer[$j]) {
                $cows++;
                $guess[$i] = "?";
                $answer[$j] = "#";
            }
        }
    }

    return ($bulls,$cows);
}
```

9. Write Perl CGI scripts to perform the following two tasks:
   (1) Produce an HTML form that allows a user to choose their favourite colour by clicking on radio buttons in a colour table. Generate the colour table from the following list of colour names

```
    @colours = ("red", "orange", "yellow",
         "green", "blue", "indigo", "violet");
```

   Assume that these are all valid names that can be used in e.g. the HTML/CSS attribute `style: "background-color: ColourName"` .

   The form should look like:

   (This may not display well when printed).

(2) Process the colour selection by printing a centered heading with the message:

Your favourite colour is *Colour*

with the word ***Colour*** replaced by the appropriate colour name *and* with the text of that word in the appropriate colour. If the user selects no colour, the script should display:

You have no favourite colour

```
#!/usr/bin/perl
use CGI ':all';
@colours = ("red", "orange", "yellow", "green", "blue", "indigo", "violet");
print(
   header(),
   start_html('Colour Chooser'),
   h3("Choose One Colour"),
   start_form(-action=>'URL to refer to processing script'),
   "<center><table border=1 cellpadding=10>\n"
);
foreach $c (@colours) {
    print(
       "<td style=\"background-color: $c\">",
       "<input type=radio name='FavouriteColour' value='$c'>",
       "</td>\n"
    );
}
print(
   "</table>\n",
   p,
   submit("I've chosen one"),
   "</center>\n",
   end_form,
   end_html
);
```

CGI.pm to process favourite colour selection:

```
#!/usr/bin/perl
use CGI ':all';
print(
   header(),
   start_html('Chosen Colour'),
);
$favColour = param('FavouriteColour');
if (!$favColour) {
    print(
       h3("You seem to have no favourite colour")
    );
}
else {
    print(
       h3("Your favourite colour is",
        "<span style=\"background-color: $favColour\">$favColour</span>"
       )
    );
}
print(end_html);
```

10. Combine the two scripts from the previous question into a single script. (Hint: you can check for the whether it's the form case or the processing case using the `param()` function with no arguments).

    Combined CGI.pm script:

```
#!/usr/bin/perl
use CGI ':all';
@colours = ("red", "orange", "yellow", "green", "blue", "indigo", "violet");
if (!param()) # no parameters => data collection case
{
    print(
       header(),
       start_html('Colour Chooser'),
       h3("Choose One Colour"),
       start_form,
       "<center><table border=1 cellpadding=10>\\n"
    );
    foreach $c (@colours) {
        print(
           "<td style=\"background-color: $c\">",
           "<input type=radio name='FavouriteColour' value='$c'>",
           "</td>\\n"
        );
    }
    print(
       "</table>\\n",
       p,
       submit("I've chosen one"),
       "</center>\\n",
       end_form,
       end_html
    );
}
else             # some parameters => data processing case
{
    print(
       header(),
       start_html('Chosen Colour'),
    );
    $favColour = param('FavouriteColour');
    if (!$favColour) {
        print(
           h3("You have have no favourite colour")
        );
    }
    else {
        print(
           h3("Your favourite colour is ",
             "<span style=\"background-color: $favColour\">$favColour</span>"
           )
        );
    }
    print(end_html);
}
```

11. Modify the script from the previous question to use a checkbox group so that users can select more than one colour. The form should look like:

I've chosen

The processing code should then print one of the following messages:

You have no favourite colour

Your favourite colour is *Colour*

Your favourite colours are $Colour_1$, $Colour_2$, ...

```
#!/usr/bin/perl
use CGI ':all';
@colours = ("red", "orange", "yellow", "green", "blue", "indigo", "violet");
if (!param()) # data collection case
{
    print(
       header(),
       start_html('Colour Chooser'),
       h3("Choose Colours"),
       start_form,
       "<center><table border=1 cellpadding=10>\\n"
    );
    foreach $c (@colours) {
        print(
           "<td style=\"background-color: $c\">",
           "<input type=checkbox name='FavouriteColours' value='$c'>",
           "</td>\\n"
        );
    }
    print(
       "</table>\\n",
       p,
       submit("I've chosen"),
       "</center>\\n",
       end_form,
       end_html
    );
}
else             # data processing case
{
    print(
       header(),
       start_html('Chosen Colours'),
    );
    @favColours = param('FavouriteColours');
    if (@favColours == 0) {
        print(
           h3("You have have no favourite colour")
        );
    }
    elsif (@favColours == 1)
    {
        print(
           h3("Your favourite colour is",
             "<span style=\"background-color: $favColours[0]\">$favColours[0]</span>"
           )
        );
    }
    else {
        foreach $c (@favColours) {
            $colourList .= "<span style=\"background-color: $c\">$c</span>, ";
        }
        $colourList =~ s/, $//;  # remove trailing ", "
        print(
           h3("Your favourite colours are",
            $colourList
           )
        );
    }
    print(end_html);
}
```