COMP[29]041 16s2 (http://www.cse.unsw.edu.au/~cs2041/16s2/)

Perl is perfect for dynamic web content

Software Construction (http://www.cse.unsw.edu.au/~cs2(

Aims

Generating dynamic web content using Perl.

Assessment

Submission: `give cs2041 lab11 guess_number.cgi guess_number.css play_guess_number.cgi [reduce.pl regex_prime.pl]`
Deadline: either during the lab, or Sunday 18 October 11:59pm (midnight)
Assessment: Make sure that you are familiar with the lab assessment criteria (lab/assessment.html).

Exercise: adding CSS to the CGI Guessing Game

In your tutorial you constructed a CGI script like this which allows the user to play a simple number guessing game, where the script picks a a number be 1 and 100 and the user guesses it.

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

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

# Simple CGI script written by andrewt@cse.unsw.edu.au
# Outputs a form which will rerun the script
# An input field of type hidden is used to pass an integer
# to successive invocations

$max_number_to_guess = 99;

print <<eof;
Content-Type: text/html

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Guess A Number</title>
</head>
<body>
eof

warningsToBrowser(1);

$number_to_guess = param('number_to_guess');
$guess = param('guess');

$game_over = 0;

if (defined $number_to_guess and defined $guess) {
    $guess =~ s/\D//g;
    $number_to_guess =~ s/\D//g;
    if ($guess == $number_to_guess) {
        print "You guessed right, it was $number_to_guess.\n";
        $game_over = 1;
    } elsif ($guess < $number_to_guess) {
        print "Its higher than $guess.\n";
    } else {
        print "Its lower than $guess.\n";
    }
} else {
    $number_to_guess = 1 + int(rand $max_number_to_guess);
    print "I've  thought of number 0..$max_number_to_guess\n";
}

if ($game_over) {
print <<eof;
    <form method="POST" action="">
        <input type="submit" value="Play Again">
    </form>
eof
} else {
print <<eof;
    <form method="POST" action="">
        <input type="textfield" name="guess">
        <input type="hidden" name="number_to_guess" value="$number_to_guess">
    </form>
eof
}

print <<eof;
</body>
</html>
eof
```

guess_number.cgi (lab/cgi/guess_number/guess_number.cgi)

I've thought of number 0..99

```
<html>
<head>
   <title>Guess A Number</title>
</head>
<body>
I've  thought of number 0..99
   <form method="POST" action="">
      <input type="textfield" name="guess">
      <input type="hidden" name="number_to_guess" value="49">
   </form>
</body>
</html>
```

Change `guess_number.cgi` to include some CSS. Put the CSS in a file named `guess_number.css`
The CSS must style at least two elements on the page.

Its not important what the CSS does - although feel free to offend your tutor's sense of aesthetics.

Hints

Look at the starting point code (assignments/matelook/matelook.cgi.txt) for assignment 2

Look at the CSS file (assignments/matelook/matelook.css) for assignment 2

These commands will get you started:

```
$ cd ~/public_html/lab11
$ chmod 755 . ..
$ chmod 755 guess_number.cgi
$ vi guess_number.cgi
$ firefox http://www.cse.unsw.edu.au/~z5555555/lab11/guess_number.cgi &
```

When you make progress on `guess_number.cgi` don't forget to push it to gitlab.cse.unsw.edu.au using the usual commands.

```
$ git add guess_number.cgi guess_number.css
$ git commit -m 'CSS being included'
$ git push
```

guess_number.cgi with CSS added:

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

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

# Simple CGI script written by andrewt@cse.unsw.edu.au
# Outputs a form which will rerun the script
# An input field of type hidden is used to pass an integer
# to successive invocations

$max_number_to_guess = 99;

print <<eof;
Content-Type: text/html

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Guess A Number</title>
    <link href="guess_number.css" rel="stylesheet">
</head>
<body>
<div class=\"message_text">
eof

warningsToBrowser(1);

$number_to_guess = param('number_to_guess');
$guess = param('guess');

$game_over = 0;

if (defined $number_to_guess and defined $guess) {
    $guess =~ s/\D//g;
    $number_to_guess =~ s/\D//g;
    if ($guess == $number_to_guess) {
        print "You guessed right, it was $number_to_guess.\n";
        $game_over = 1;
    } elsif ($guess < $number_to_guess) {
        print "Its higher than $guess.\n";
    } else {
        print "Its lower than $guess.\n";
    }
} else {
    $number_to_guess = 1 + int(rand $max_number_to_guess);
    print "I've  thought of number 0..$max_number_to_guess\n";
}

print "</div>\n";

if ($game_over) {
print <<eof;
    <form method="POST" action="">
        <input type="submit" value="Play Again" class="play_again_button">
    </form>
eof
} else {
print <<eof;
    <form method="POST" action="">
        <input type="textfield" name="guess" class="input_number">
        <input type="hidden" name="number_to_guess" value="$number_to_guess">
    </form>
eof
}

print <<eof;
</body>
</html>
eof
```

Some not very useful CSS stolen from the assignment:

```
.message_text {
    padding-top: 1em;
    padding-bottom: 1em;
    font-size: x-large;
    font-weight: bold;
}

.input_number {
    background-color: #ABCDEF;
    color: #204142;
    border:thin solid #204142;
    border-radius: 1em;
    padding-left: 0.42em;
}

.play_again_button {
    background-color: #FEDBCA;
    border:thin solid #904142;
    border-radius: 0.42em;
    color: #904142;
}
```

Exercise: Let the CGI Script Guess

Write a Perl CGI script which instead plays the number guessing game. The user chooses a number between 1 and 100 and then the script attempts to gu

Your script should use an optimal guessing strategy (simple binary search).

Here is an example implementation:

play_guess_number.cgi (lab/cgi/guess_number/play_guess_number.cgi)

My guess is: 50 Higher? Correct? Lower?

```
<html>
<head>
   <title>A Guessing Game Player</title>
</head>
<body>
   <form method="POST" action="">
      My guess is: 50
      <input type="submit" name="higher" value="Higher?">
      <input type="submit" name="correct" value="Correct?">
      <input type="submit" name="lower" value="Lower?">
      <input type="hidden" name="low_limit" value="1">
      <input type="hidden" name="high_limit" value="100">
      <input type="hidden" name="guess" value="50">
```

Hints

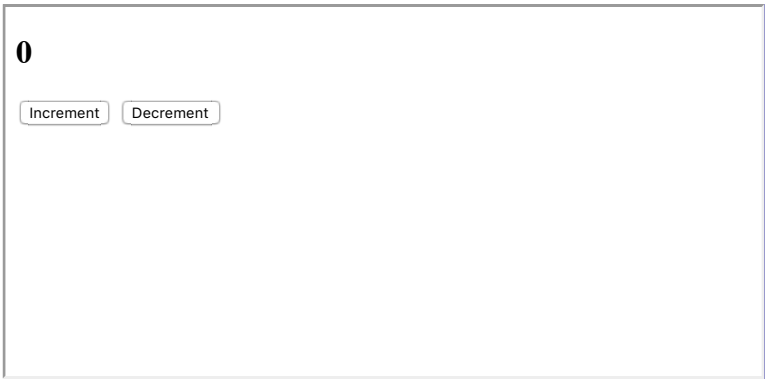
Use 3 hidden variables to store upper and lower bounds and your last guess.

Look at the CGI examples from lectures (http://cgi.cse.unsw.edu.au/~cs2041cgi/code/cgi/index.html) particularly the Handling multiple submit buttons e (http://cgi.cse.unsw.edu.au/~cs2041cgi/code/cgi/multiple_submit_buttons.cgipm.cgi) (src (http://cgi.cse.unsw.edu.au/~cs2041cgi/code/cgi/multiple_submit_buttons.cgipm++.cgi.txt))

Note in the CGI examples (lab/cgi/code/cgi/cgi_examples.html) from lectures you have been shown a CGI script with multiple submit buttons in a form:

```
#!/usr/bin/perl -w
# Simple CGI script written by andrewt@cse.unsw.edu.au
# Outputs a form which will rerun the script
# An input field of type hidden is used to pass an integer
# to successive invocations
# Two submit buttons are used to produce different actions

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

print <<eof;
Content-Type: text/html

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Handling Multiple Submit Buttons</title>
</head>
<body>
eof
warningsToBrowser(1);

$hidden_variable = param("x") || 0;

if (defined param("increment")) {
        $hidden_variable++;
} elsif (defined param("decrement")) {
        $hidden_variable--;
}

print <<eof;
<h2>$hidden_variable</h2>
<form method="post" action="">
<input type=hidden name="x" value="$hidden_variable">
<input type="submit" name="increment" value="Increment">
<input type="submit" name="decrement" value="Decrement">
</form>
</body>
</html>
eof
```

multiple_submit_buttons.cgipm.cgi (lab/cgi/guess_number/multiple_submit_buttons.cgipm.cgi)




```
<html>
<head>
   <title>Handling Multiple Submit Buttons</title>
</head>
<body>
<h2>0</h2>
<form method="post" action="">
<input type="hidden" name="x" value="0">
<input type="submit" name="increment" value="Increment">
<input type="submit" name="decrement" value="Decrement">
</form>
</body>
</html>
```

You can get started similar to last week, so if you want to use the example CGI script from lectures as a starting point you might do this:

```
$ cd ~/public_html/lab11
$ cp -p guess_number.cgi play_guess_number.cgi
$ chmod 755 .
$ chmod 755 play_guess_number.cgi
$ vi play_guess_number.cgi
$ firefox http://www.cse.unsw.edu.au/~z5555555/lab11/play_guess_number.cgi &
```

When you make progress on `play_guess_number.cgi` don't forget to push it to gitlab.cse.unsw.edu.au using the usual commands.

```
$ git add play_guess_number.cgi
$ git commit -m 'first part implemented'
$ git push
```

Sample solution for play_guess_number.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
# Script tries to guess a number between 1 and 100

print <<eof;
Content-Type: text/html

<!DOCTYPE html>
<html lang="en">
<head>
    <title>A Guessing Game Player</title>
</head>
<body>
eof

warningsToBrowser(1);

# we guessed right
if (param('correct')) {
    print <<eof;
    <form method="POST" action="">
        I win!!!!
       <input type="submit" value="Play Again">
    </form>
</body>
</html>
eof
    exit 0;
}


$low_limit = param('low_limit');
$high_limit = param('high_limit');
$last_guess = param('guess');

# limit missing so we must be starting a new game

if (!defined $low_limit || !defined $high_limit) {
    $low_limit = 1;
    $high_limit = 100;
} elsif (defined $last_guess) {
    if (defined param('lower')) {
        $high_limit = $last_guess - 1;
    } elsif (defined param('higher')) {
        $low_limit = $last_guess + 1;
    }
}

$guess = int (($low_limit + $high_limit)/2);

print <<eof;
    <form method="POST" action="">
        My guess is: $guess
        <input type="submit" name="higher" value="Higher?">
        <input type="submit" name="correct" value="Correct?">
        <input type="submit" name="lower" value="Lower?">
        <input type="hidden" name="low_limit" value="$low_limit">
        <input type="hidden" name="high_limit" value="$high_limit">
        <input type="hidden" name="guess" value="$guess">
    </form>
</body>
</html>
eof
```

Challenge Exercise: Perl Reduce

Write a Perl function `reduce` which takes two arguments a reference to some Perl code and a list. It should reduce the list to a single value by calling the setting the variables $a and $b each time. The first call will be with $a and $b set to the first two elements of the list, subsequent calls will be done by set to the result of the previous call and $b to the next element in the list.
If the list is empty then undef should be returned, if the only contains one element then should be returned with out the code being executed.

A number of common list operations can be defined in terms of reduce. For example:

```
$sum = reduce { $a + $b } 1 .. 10;
$min = reduce { $a < $b ? $a : $b } 5..10;
$maxstr = reduce { $a gt $b ? $a : $b } 'aa'..'ee';
$concat = reduce { $a . $b } 'J'..'P';
$sep = '-';
$join = reduce { "$a$sep$b" }  'A'..'E';
print "$sum $min $maxstr $concat $join\n";
```

should print:

```
55 5 ee JKLMNOP A-B-C-D-E
```

Hint, you can find a suitable prototype for reduce by looking at the quicksort examples from lectures. Without a suitable prototype you'll need to call redu slightly more clumsy manner:

```
$sum = reduce sub { $a + $b }, 1 .. 10;
$min = reduce sub { $a < $b ? $a : $b }, 5..10;
$maxstr = reduce sub { $a gt $b ? $a : $b }, 'aa'..'ee';
$concat = reduce sub { $a . $b }, 'J'..'P';
$sep = '-';
$join = reduce sub { "$a$sep$b" }, 'A'..'E';
print "$sum $min $maxstr $concat $join\n";
```

You are not permitted to use `List::Util` which contains a reduce function.

Don't look for other people solutions - see if you can come up with your own.

Sample solution for reduce.pl

```
#!/usr/bin/perl -w
# simple implementation of reduce
# List::util has a more robust implementation
sub reduce(&@) {
    my ($code, @list) = @_;
    return undef if !@list;
    local $a = shift @list;
    while (@list) {
        local $b = shift @list;
        $a = &$code;
    }
    return $a;
}
$sum = reduce { $a + $b } 1 .. 10;
$min = reduce { $a < $b ? $a : $b } 5..10;
$maxstr = reduce { $a gt $b ? $a : $b } 'aa'..'ee';
$concat = reduce { $a . $b } 'J'..'P';
$sep = '-';
$join = reduce { "$a$sep$b" }  'A'..'E';
print "$sum $min $maxstr $concat $join\n";
```

Challenge Exercise: Unary Reduce

Write a Perl or Python regex which matches unary number iff it is composite (not prime).

In other words write a regex that matches a string of *n* ones iff *n* is composite.

For example this Perl:

```
$regex = ?;
foreach $n (1..100) {
    $unary = 1 x $n;
    print "$n = $unary unary - ";
    if ($unary =~ $regex) {
        print "composite\n"
    } else {
        print "prime\n";
    }
}
```

with the regex added should print:

```
1 = 1 unary - composite
2 = 11 unary - prime
3 = 111 unary - prime
4 = 1111 unary - composite
5 = 11111 unary - prime
6 = 111111 unary - composite
7 = 1111111 unary - prime
8 = 11111111 unary - composite
9 = 111111111 unary - composite
10 = 1111111111 unary - composite
11 = 11111111111 unary - prime
12 = 111111111111 unary - composite
13 = 1111111111111 unary - prime
....
```

Hint: you can't do this with a true regular expression, i.e using `|*()` alone, you need to use features that Perl and Python add.

Don't google for other people solutions - see if you can come up with your own.

Sample solution for regex_prime.pl

```
#!/usr/bin/perl -w
# regex widely repeated - apparently originally by http://www.abigail.be/
$regex = '^1?$|^(11+?)\1+$';
foreach $n (1..100) {
    $unary = 1 x $n;
    print "$n = $unary unary - ";
    if ($unary =~ $regex) {
        print "composite\n"
    } else {
        print "prime\n";
    }
}
```

Finalising

You must show your solutions to your tutor and be able to explain how they work. Once your tutor has discussed your answers with you, you should subm them using:

```
$ give cs2041 lab11 guess_number.cgi guess_number.css play_guess_number.cgi [reduce.pl regex_prime.pl]
```

Whether you discuss your solutions with your tutor this week or next week, you must submit them before the above deadline.