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/~cs2041/16s2/)

Aims

Generating dynamic web content using Perl.

Assessment

Submission: `give cs2041 lab10 creditcard.pl creditcard.cgi testing_plan.txt [quine.pl quinepy.pl quine.pypl]`
Deadline: either during the lab, or Monday 10 October 11:59pm (midnight)
Assessment: Make sure that you are familiar with the lab assessment criteria (lab/assessment.html).

Validating a Credit Card Number in a Perl Application

In the tutorial you discussed this Python code which implements the Luhn formula (https://en.wikipedia.org/wiki/Luhn_algorithm) which credit card numbers must satisfy.

```
#!/usr/bin/python
# written by andrewt@cse.unsw.edu.au as a COMP2041 programming example
# validate a credit card number by calculating its
# checksum using Luhn's formula (https://en.wikipedia.org/wiki/Luhn_algorithm)

import re, sys

def luhn_checksum(number):
    checksum = 0
    digits = reversed(number)
    for (index, digit) in enumerate(digits):
        multiplier = 1 + index % 2
        d = int(digit) * multiplier
        if d > 9:
            d -= 9
        checksum += d
    return checksum

def validate(credit_card):
    number = re.sub(r'\D', '', credit_card)
    if len(number) != 16:
        return credit_card + " is invalid - does not contain exactly 16 digits"
    elif luhn_checksum(number) % 10 == 0:
        return credit_card + " is valid"
    else:
        return credit_card + " is invalid"

if __name__ == "__main__":
    for arg in sys.argv[1:]:
        print(validate(arg))
```

Write a Perl script `creditcard.pl` which performs the same task. For example:

```
$ creditcard.pl 0000000000000000 9999999999999999
0000000000000000 is valid
9999999999999999 is invalid
$ creditcard.pl 2389423423423467 9182387723427777 9182380923427773 4564456445644564
2389423423423467 is valid
9182387723427777 is invalid
9182380923427773 is invalid
4564456445644564 is valid
$ creditcard.pl 42 ................
42 is invalid - does not contain exactly 16 digits
................ is invalid - does not contain exactly 16 digits
$ creditcard.pl 4564-7953-6021-9047 1234-5678-9012-3456
4564-7953-6021-9047 is valid
1234-5678-9012-3456 is invalid
```

Hints

Split `split //, $string` returns a list of the characters in a string.
The builtin Perl function reverse returns a list in reversed order.

Keep the same functions as the Python.

```
$ cd ~/public_html/lab10
$ vi creditcard.pl
. . . .
```

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

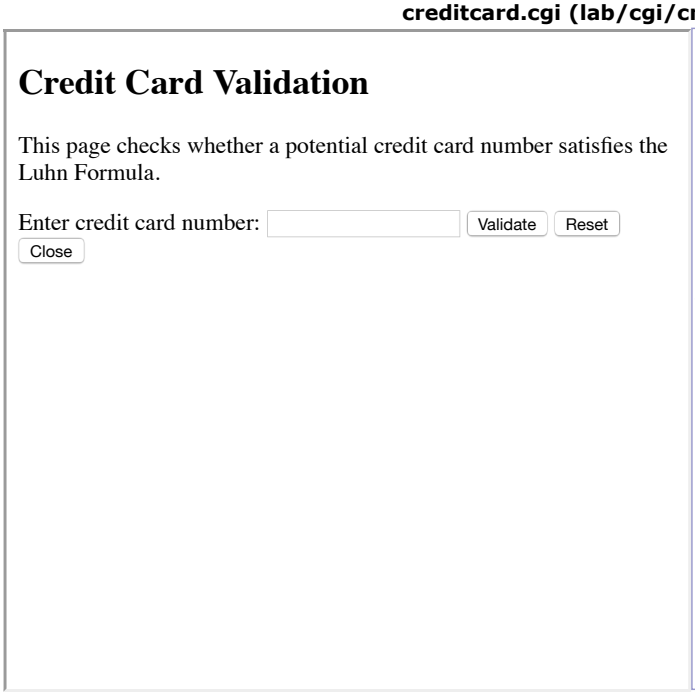
```
$ git add creditcard.pl
$ git commit -m 'checksum calculated correctly'
$ git push
```

Validating a Credit Card Number in a Perl CGI Script

Write a Perl CGI script `creditcard.cgi` which embedded your code to validate a credit card number in a CGI script.

Match EXACTLY the behaviour of this example implementation:

creditcard.cgi (lab/cgi/creditcard/creditcard.cgi)




```
<html>
<head>
<title>Credit Card Validation</title>
<meta http-equiv="Content-Type" content="text/html; charset=
</head>
<body>

<h2>Credit Card Validation</h2>
This page checks whether a potential credit card number satisfies
<p>
</p><form>
Enter credit card number:
<input type="text" name="credit_card" value="">
<input type="submit" name="submit" value="Validate">
<input type="reset" name="Reset" value="Reset">
<input type="submit" name="close" value="Close">
</form>
</body>
</html>
```

Hints

Copy your code for `creditcard.pl` to `creditcard.cgi`.

CSE's web server is configured to treat files sending with this suffix as cgi scripts and will execute them when they are requested.

```
$ cp creditcard.pl  creditcard.cgi
$ chmod 755 creditcard.cgi
$ firefox http://cgi.cse.unsw.edu.au/~z5555555/lab10/creditcard.cgi &
```

Remember if you are not working in a CSE lab run firefox (or another web browser) on your local machine.

For example if you are working at home on a laptop and using ssh to connect to CSE, run the web browser on your laptop and supply the URL `http://cgi.cse.unsw.edu.au/~z5555555/lab09/browser.cgi`.

Simply renaming it doesn't change `credit.pl` from a perl program into a useful CGI script. It also needs to be modified to collect the credit card as a parameter and to produce HTML rather than plain text.

Let's start by doing a quick hack to get something appearing in your Web browser, just to make sure that you've got the script set up right.

Add this to the top of `creditcard.cgi`:

```
use CGI qw/:all/;
use CGI::Carp qw/fatalsToBrowser warningsToBrowser/;

print header, start_html("Credit Card Validation"), "\n";
warningsToBrowser(1);
$credit_card = param("credit_card");
if (defined $credit_card) {
    print validate($credit_card);
}
print end_html;
exit 0;
```

Note this code assumes you have a Perl *validate* function equivalent the one in the above Python.

Run the CGI script from the command line to make sure it produces appropriate output:

```
$ ./creditcard.cgi
Content-Type: text/html; charset=ISO-8859-1

<!DOCTYPE html
...
```

Now try running your code as your CGI script, you can supply the credit card number as a query string by using a URL like this

```
http://cgi.cse.unsw.edu.au/~z5555555/lab10/creditcard.cgi?credit_card=4564-7953-6021-9047
```

When that works you now have to implement the full logic of the CGI script so it matches the reference implementation.

There are four kinds of pages to be delivered by your script (although they have common elements):

1. A basic "Welcome" page with a brief message, a prompt next to a text box to accept a [hypothetical] credit card number, and three buttons, labelled "Validate", "Reset" and "Close". The text box is initially empty.
2. A successful report page, which looks the same as the basic page but has a message indicating what the input was and that it was valid (assuming it passes the validation tests). The entry box is cleared ready for a new number, and the prompt is changed to "Another card number:" or similar.
3. An error report page, which looks the same as the basic page but has an error message (wrong number of digits and invalid number are the error conditions detected). In this case the previous (erroneous) input remains in the text field, and the prompt is changed to "Try again:".
4. A final "Goodbye" page, which contains only a finalisation message, and is presented in response to pressing the Close button.

What do I do about Internal Server Error?

If your script is producing a 500 error from the webserver you can obtain some debugging info by creating a `.htaccess` file with these contents:

```
<Files "creditcard.cgi">
SetHandler application/x-setuid-cgid
</Files>
```

See here (http://taggi.cse.unsw.edu.au/FAQ/CGI_scripts/) for more info.

Cross-site Scripting

Try pasting this HTML into the credit card field of your CGI script and validating it

```
<img src="http://imgs.xkcd.com/comics/exploits_of_a_mom.png">
```

If your CGI script displays an image your script allows cross-site-scripting (XSS). This is a serious security vulnerability, even if this particular script is unimportant. For example, it may allow a malicious attacker to steal user's cookies for your site.

You should never output HTML supplied by a user without sanitizing it.

A similar risk applies to field values (even if hidden!) - try entering a value containing a double-quote(") and see what your script does, For example this string:

```
"> <img src="http://imgs.xkcd.com/comics/exploits_of_a_mom.png">
```

Systematic Testing

Create a text file `testing_plan.txt` containg a list of tests you've done to compare the reference implementation

Make sure your test plan explores all user pathways.

Remember that any button can be pushed, and any text typed into a text field: your script must cover all possibilities.

- test name
- value in the text field
- button pushed
- expected response
- actual response (if different)

Challenge Exercise: A Perl program that Outputs Itself

Write a (non-empty) Perl program `quine.pl` which takes no input or arguments and when run outputs itself (its own source code).

For example:

```
$ perl quine.pl >output
$ diff output quine.pl
$ <quine.pl perl|perl|perl|perl >output
$ diff output quine.pl
$
```

You are not permitted to use the DATA file stream, access the source file, or use any Perl builtin or module which lets a Perl program access its source.

Don't look for other people solutions.

See if you can invent your own - there are many possible approaches.

Tutors will be generous for original but not-quite working attempts. Tutors won't give marks to non-original solutions.

Challenge Exercise: A Perl program that Outputs a Python program that Outputs the Perl Program

More difficult: write a (non-empty) Perl program `quinepy.pl` which when run outputs a Python program which when run outputs the Perl program. For example:

```
$ perl quinepy.pl|python >output
$ diff output quinepy.pl
$ <quinepy.pl perl|python|perl|python|perl|python >output
$ diff output quinepy.pl
$
```

Challenge Exercise: A Perl & Python program that Outputs itlsef

Even more difficult: write a (non-empty) program `quine.pypl` which is both valid Perl & Python and when run with either outputs itself. For example:

```
$ perl quine.pypl >output
$ diff output quine.pypl
$ python quine.pypl >output
$ diff output quine.pypl
$ <quine.pypl perl|python|python|perl|perl|python >output
$ diff output quine.pypl
$
```

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 submit them using:

```
$ give cs2041 lab10 creditcard.pl creditcard.cgi testing_plan.txt [quine.pl quinepy.pl quine.pypl]
```

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