Week 08 Laboratory Sample Solutions

Objectives

- Developing Perl & Shell skills
- Exploring simple approaches to scraping data from the web

Preparation

Before the lab you should re-read the relevant lecture slides and their accompanying examples.

Getting Started

Create a new directory for this lab called `lab08`, change to this directory, and fetch the provided code for this week by running these commands:

```
$ mkdir lab08
$ cd lab08
$ 2041 fetch lab08
```

Or, if you're not working on CSE, you can download the provided code as a zip file or a tar file.

EXERCISE:

What Courses Does UNSW Have this Year - Shell

Write a Shell script `courses.sh` which given a course prefix, e.g. COMP, prints all the course code and name for all UNSW shell_courses with that prefix offered this year on the Kensington Campus. For example:

```
$ ./courses.sh VISN
VISN1101 Seeing the World: Perspectives from Vision Science
VISN1111 Geometrical and Physical Optics
VISN1221 Visual Optics
VISN2111 Ocular Anatomy and Physiology
VISN2211 Organisation and Function of the Visual System
VISN3111 Development and Aging of the Visual System
VISN4016 Vision Science Honours
$ ./courses.sh COMP|tail
COMP9511 Human Computer Interaction
COMP9517 Computer Vision
COMP9596 Research Project
COMP9801 Extended Design and Analysis of Algorithms
COMP9814 Extended Artificial Intelligence
COMP9844 Extended Neural Networks and Deep Learning
COMP9900 Information Technology Project
COMP9991 Research Project A
COMP9992 Research Project B
COMP9993 Research Project C
```

Your program must be POSIX compatible Shell. You can assume anything that works with the version of `/bin/dash` on CSE systems is POSIX compatible.

You are not permitted to use other languages such as Perl, Python, C, ...

You are permitted to use programs such as egrep, sed, sort, uniq.

Hints:

The information you need for course code prefix `COMP` can be found in this web page:
http://www.timetable.unsw.edu.au/current/COMPKENS.html. You can assume this is the case for all prefixes.

The command `curl` will download a URL and print it to standard output.

```
$ curl --location --silent http://www.timetable.unsw.edu.au/current/COMPKENS.html|head
<title>Class Search by Teaching Period</title>
<link rel="stylesheet" type="text/css" href="../layout/2020/myunsw.css">
<head>
      <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">

   <table width="100%" cellspacing="0" cellpadding="0">
      <form name="googleForm" method="GET" action="http://www.google.com/u/theuniversityofnewsouthwales" target="_blank">
         <tr>
            <td width="30%" style="height:120px; border-bottom:10px solid #FFCC00; background-color:#fff;"><a
href="http://www.unsw.edu.au" target="_blank"><img border="0" src="/images-timetable/banner2020.jpg" alt="The University
of New South Wales" width="836" height="179" style="float:left; margin-left:20px; margin-top:25px;"></a></td>
            <td width="70%" style="height:120px; border-bottom:10px solid #FFCC00; background-color:#fff; vertical-
align:bottom; " align="right">
$
```

In a script it is best run as `curl --silent` so it doesn't print extra information on standard error.

The `--location` is required so `curl` will follow a HTTP redirect from the URL.

The `wget -q -O-` could also be used.

You may find uniq's **-w** option useful when removing duplicate courses.

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest shell_courses
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs2041 lab08_shell_courses courses.sh
```

before **Tuesday 28 July 21:00** to obtain the marks for this lab exercise.

Sample solution for `courses.sh`

```
#!/bin/dash

for course_prefix in "$@"
do
    url="http://www.timetable.unsw.edu.au/current/${course_prefix}KENS.html"

    # could also use wget --quiet -O-

    curl --location --silent "$url"|
    egrep "$course_prefix[0-9]{4}\.html"|
    sed '
        s/.*href="//
        s/.html">/ /
        s/<.*//
        '|
    egrep -v "$course_prefix[0-9]{4} $course_prefix[0-9]{4}$"|
    sort|
    uniq -w8

done
```

EXERCISE:

What Courses Does UNSW Have this Year - Perl

Write a Perl script `courses.pl` which given a course prefix, e.g. COMP, prints all the course code and name for all UNSW courses with that prefix offered this year on the Kensington Campus. For example:

```
./courses.pl VISN
VISN1101 Seeing the World: Perspectives from Vision Science
VISN1111 Geometrical and Physical Optics
VISN1221 Visual Optics
VISN2111 Ocular Anatomy and Physiology
VISN2211 Organisation and Function of the Visual System
VISN3111 Development and Aging of the Visual System
VISN4016 Vision Science Honours
$ ./courses.pl COMP|tail
COMP9511 Human Computer Interaction
COMP9517 Computer Vision
COMP9596 Research Project
COMP9801 Extended Design and Analysis of Algorithms
COMP9814 Extended Artificial Intelligence
COMP9844 Extended Neural Networks and Deep Learning
COMP9900 Information Technology Project
COMP9991 Research Project A
COMP9992 Research Project B
COMP9993 Research Project C
```

Your answer must be Perl only.

You may not run external programs, e.g. via system or backquotes.

Hints:

The information you need for course code prefix `COMP` can be found in this web page: http://www.timetable.unsw.edu.au/current/COMPKENS.html. You can assume this is the case for all prefixes.

The Perl module **LWP::Simple** provides a very simple way to get a web page, e.g:

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

use LWP::Simple;
$url = "http://www.timetable.unsw.edu.au/current/COMPKENS.html";
$web_page = get($url) or die "Unable to get $url";
print $web_page;
```

You will need to remove duplicate entries for some courses.

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest perl_courses
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs2041 lab08_perl_courses courses.pl
```

before **Tuesday 28 July 21:00** to obtain the marks for this lab exercise.

Sample solution for `courses.pl`

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

use LWP::Simple;

foreach $course_prefix (@ARGV) {
    my $url = "http://www.timetable.unsw.edu.au/current/${course_prefix}KENS.html";

    my $web_page = get($url) or die "Unable to get $url";

    my @course_links = $web_page =~ /<a href="$course_prefix\d{4}\.html">.*?<\/a>/g;

    my %courses;
    foreach $link (@course_links) {
        my ($course_code, $course_name) = $link =~ /($course_prefix\d{4})\.html">(.*?)</ or die;
        next if $course_name eq $course_code;
        $courses{$course_code} = $course_name;
    }

    foreach $course_code (sort keys %courses) {
        print "$course_code $courses{$course_code}\n";
    }
}
```

Alternative solution for `courses.pl`

```
#!/usr/bin/env perl
# courses.pl --- find the courses on offer at UNSW
#
# 2020-07-24    Jashank Jeremy <jashank.jeremy@unsw.edu.au>
#     coded live in fri14b, when someone asked for "the right way"

=pod

There's a substantial amount of inline exposition, written in the Perl
"Plain Old Documentation" format.  You may wish to have a quick peek at
the L<perlpod> entry in the Perl documentation collection, to get a feel
for what's going on.

=cut

use strict;
use warnings;

use LWP::Simple;
use XML::LibXML;

foreach my $prefix (@ARGV) {
    # Construct the URL, and retrieve contents.
    my $url = sprintf
        'http://www.timetable.unsw.edu.au/current/%sKENS.html',
        $prefix;
    my $content = get($url) or die "Unable to get $url";

=pod

We invoke a real HTML and XML parser --- in this case, C<libxml2>, which
we load via L<XML::LibXML>, available on CPAN --- but this document is
so malformed that libxml2 can't cope and fails to parse without
assistance.  This is a regrettably common issue with HTML; a "tag-soup"
parser might be a better choice here.

As an aside on general vs. bespoke solutions, this specific task might
be more easily done with a regular-expression-based solution: in this
case, the HTML is a *huge* mess, and our general-case tool cannot cope.
A bespoke tool for this specific task might be a better approach.

First, we must "fix" the malformed HTML: there's a huge slab of weird
cruft before the opening C<< <html> >> tag, which we'll just throw away.
C</m> makes C<^> and C<$> match individual lines within the string.
C</s> changes C<.> to match newlines.
See L<perlre> for more details.

Then, we pass this string into libxml2, which provides us an object that
we can query at our leisure.

=cut

    $content =~ s/.*<html/<html/ms;
    my $doc = XML::LibXML->load_html(
        string   => $content,
        recovery => 2
    );

=pod

One of the advantages of using "the right tools" here is that we can
rely on document structure, and find specific things within it by its
landmarks.  Working out what we want to find can be done interactively
using, e.g., a normal web browser's document inspector.

Examining our document, we find table rows of three columns ---

    | course-code | course-name | units-of-credit |

which gives us some indicative structure.  We track down all the table
data cells, elements of type C<TD> which here have class C<data>.  These
```

```
all contain an element C<A>.  In XPath --- a common syntax for referring
to paths within an XML document, which is well-specified elsewhere ---
we could phrase this with the expression

    //td[@class="data"]/a

libxml2, as with most XML parsers, can natively execute XPath queries
and produce a list of the matching terms; we do this to the document.
This gives us a list of all the elements; which so happen to come in
pairs --- for example,

    [
        ...,
        <a href="COMP2041.html">COMP2041</a>,
        <a href="COMP2041.html">Software Construction: Techniques and Tools</a>,
        ...,
    ]

This also happens to catch some other odd table rows, so we filter this
to only C<A> elements whose hyperlink reference (C<href>) attribute
looks like a course code.  We could probably do this with XPath, but I'm
too lazy to go look up the specific syntax to do that.  And in any case,
we've got Perl's matching facilities available; we may as well use them.

    $_->getAttribute("href") =~ /\A[A-Z]{4}[0-9]{4}/

We use C<grep> over the list of elements for its filtering effect.
Remember that C<grep> places each element in C<$_>, and then evaluates
the block.

Given a reference, the C<< -> >> operator allows us to operate on the
data within it, somewhat like in C.  In Perl, that might be a hashref,
which allows us to retrieve by key from the hash; or an arrayref which
allows us to retrieve by index --- or, here, a blessed reference: a
reference which has a package associated with it.  We can invoke the
subroutines in that package, passing the referenced value as its first
argument.  See C<bless> in L<perlfunc>, and L<perlobj> for more.

Here, C<$_> is an XML element, which has some subroutines associated ---
for example, C<getAttribute>, which unsurprisingly retrieves a specific
attribute's value.

We still have XML elements, though; but their text is now what we're
interested in; we can use the C<nonBlankChildNodes> subroutine to
retrieve that.  Now we have an array of course codes and names ---

    [
        ...,
        "COMP2041",
        "Software Construction: Techniques and Tools",
        ...,
    ]

--- which we can just assign into a hash to examine at our leisure.

This all rolls up into the delightfully terse:

=cut

    my %courses =
      map  { $_->nonBlankChildNodes() }
      grep { $_->getAttribute("href") =~ /\A[A-Z]{4}[0-9]{4}/ }
      $doc->findnodes('//td[@class="data"]/a');

    foreach my $code (sort keys %courses) {
        print "$code $courses{$code}\n";
    }
}
```

CHALLENGE EXERCISE:

When Do These Courses Have Lectures

Write a Perl script `timetable.pl` which given course codes prints an ASCII timetable of their lecture times for this year in the format shown in the example below.

```
$ ./timetable.pl COMP2041 MATH2400 MATH2859
         Mon   Tue   Wed   Thu   Fri
09:00     L                       L
10:00     L                       L
11:00           L           L     L
12:00           L
13:00
14:00           L
15:00           L
16:00
17:00
18:00
19:00
20:00
$ ./timetable.pl COMP9044 COMP6771 GSOE9820
        Mon   Tue   Wed   Thu   Fri
09:00
10:00                         L
11:00                         L
12:00
13:00               L
14:00           L   L
15:00           L
16:00                     L
17:00                     L
18:00                     L
19:00                     L
20:00
```

You can assume that a course's lecture times will be found in a web page equivalent to: http://timetable.unsw.edu.au/current/MATH1131.html.

Hint: if you find it difficult to use a regex to extract the line containing the lecture description, split the page into line match a previous line, then skip a certain number of lines.

You assume there are no lectures on weekends, and no lectures before 09:00 or after 21:00

Your answer must be Perl only.

You may not run external programs, e.g. via system or backquotes.

If a course is running in multiple terms print all the terms.

If there are multiple streams print all the streams, for example:

```
$ ./timetable.pl COMP1511
         Mon    Tue    Wed    Thu    Fri
09:00            L
10:00            L
11:00                   L
12:00                   L
13:00
14:00
15:00
16:00                          L
17:00                          L
18:00
19:00
20:00
$ ./timetable.pl MATH1131
        Mon    Tue    Wed    Thu    Fri
09:00            L
10:00            L
11:00
12:00                   L
13:00      L     L     L
14:00      L     L            L      L
15:00      L     L            L      L
16:00      L            L     L
17:00            L
18:00
19:00
20:00
```

Note the correct output in all the above examples will change when next term's lectures are added to the timetable.

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest timetable
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs2041 lab08_timetable timetable.pl
```

before **Tuesday 28 July 21:00** to obtain the marks for this lab exercise.

Sample solution for `timetable.pl`

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

use LWP::Simple;

$base_url = "http://timetable.unsw.edu.au/current";
@possible_lecture_days = qw/Mon Tue Wed Thu Fri/;
@possible_lecture_times = (9..20);
$column_width = 6;

foreach $course (@ARGV) {
    my $url = "$base_url/$course.html";
    my $web_page = get($url) or die "Unable to get $url";
    my @lines = split "\n", $web_page;
    while (@lines) {
        $line = shift @lines;
        next if $line !~ /href.*>Lecture/;

        @lines = @lines[5..$#lines]; # skip 5 lines
        $line = shift @lines;

        $line =~ s/<.*?>//g; # remove tags

        foreach $lecture (split /\), /, $line) {
             my @days = $lecture =~ /\b([a-z]{3})\b/gi ;

            my ($start_hour,$finish_hour,$finish_minute) = $lecture =~ /(\d\d):\d\d - (\d\d):(\d\d)/ or next;
            $finish_hour++ if $finish_minute ne "00";

            foreach $day (@days) {
                foreach $time ($start_hour..$finish_hour-1) {
                    $lecture{$day}{$time}++
                }
            }
        }
    }
}

printf "%${column_width}s", $_ foreach ('', @possible_lecture_days);
print "\n";
foreach $time (@possible_lecture_times) {
    printf "%02d:00", $time;
    foreach $day (@possible_lecture_days) {
        printf "%${column_width}s", $lecture{$day}{$time} ? "L" : " ";
    }
    print "\n";
}
```

Submission

When you are finished each exercises make sure you submit your work by running `give`.

You can run `give` multiple times. Only your last submission will be marked.

Don't submit any exercises you haven't attempted.

If you are working at home, you may find it more convenient to upload your work via give's web interface.

Remember you have until **Tuesday 28 July 21:00** to submit your work.

You cannot obtain marks by e-mailing your code to tutors or lecturers.

You check the files you have submitted here.

Automarking will be run by the lecturer several days after the submission deadline, using test cases different to those `autotest` runs for you. (Hint: do your own testing as well as running `autotest`.)

After automarking is run by the lecturer you can view your results here. The resulting mark will also be available via give's web interface.

Lab Marks

When all components of a lab are automarked you should be able to view the the marks via give's web interface or by running this command on a CSE machine:

```
$ 2041 classrun -sturec
```
