INGM BFG master - Practical 2

Perl (2)

Introduction

- This practical contains further exercises that are intended to familiarise you with Perl Programming. While you work through the tasks below compare your results with those of your fellow students and ask for help and comments if required.
- This document can be found at

https://homes.di.unimi.it/re/Corsi/INGM_master_BFG/PERLCOMPBIO_practical2.pdf

  and you might proceed more quickly if you cut-and-paste code from that PDF file. Note that a cut-and-paste operation may introduce extra spaces into your code. It is important that those are removed and that your code exactly matches that shown in this worksheet.
- The exercises and instructions in this worksheet assume that you use the Department's Linux systems to experiment with Perl.

  If you want to use the Department's Windows systems and our Perl installation on Windows instead, then you can do so.
- To keep things simple, we will just use a text editor and a terminal. You can use whatever text editor you are most familiar or comfortable with.
- If you do not manage to get through all the exercises during this practical session, please complete them in your own time before the next practical takes place.

Exercises

1. Let us start with the introductory example for the use of regular expressions that you have seen in the lectures.

   a. Open a text editor and enter the following Perl code:

```
#!/usr/bin/perl
# Author: <your name>
# Practical 2: Perl Pattern Matching and Substitution [perl02A]

$log1 = "Generating an unsorted array took 1.259 seconds\n";
$log1 .= "Sorting took 10.486 seconds\n";
$log1 .= " Generating an unsorted array took 1.346 seconds\n";
$log1 .= " Sorting took 9.276 seconds\n";
print $log1;
```

   Replace `<your name>` with your own name.

   b. Save the code to a file named `perl02A` in some appropriate directory.

   c. In a terminal, go to the directory in which the file has been stored.

   d. Make sure that the file `perl02A` is executable by using the command

```
chmod u+x,og-rwx perl02A
```

   Remember that you only have to do so once. When you later save the file again after making modifications, the file permissions will be preserved and the file remains executable.

e. Now execute the Perl script using the command
`./perl02A`
and check that the output is:

```
Generating an unsorted array took 1.259 seconds
Sorting took 10.486 seconds
 Generating an unsorted array took 1.346 seconds
 Sorting took 9.276 seconds
```

2. In the lectures we have looked at the use of regular expressions to extract information.

   a. Add the following code at the end of your Perl script, save the file and then execute it.

```
$_ = $log1;
(/Sorting took (\d+\.\d+) seconds/) && do {
  $runtime += $1;
  $count++;
  print "1: Match found: $1 -- $runtime -- $count\n";
};
print "\n";
```

   b. Check that the additional output is

```
1: Match found: 10.486 -- 10.486 -- 1
```

   c. Refer to
   http://perldoc.perl.org/functions/do.html
   to understand what the function of `do` is.

3. In the last example, we found only one match. Let us try to find all matches.

   a. Add the following code at the end of your Perl script, save the file and then execute it.

```
$runtime = $count = 0;
$_ = $log1;
while (/Sorting took (\d+\.\d+) seconds/ && $i++ < 10) {
  $runtime += $1;
  $count++;
  print "2: Match found: $1 -- $runtime -- $count\n";
};
print "\n";
```

   b. As you can see from the output, the code does not work as desired, it finds the same match again and again.
   Correct the code by adding the appropriate modifier to `/ /`. Execute the corrected code. If your code is correct, the output will be

```
2: Match found: 10.486 -- 10.486 -- 1
2: Match found: 9.276 -- 19.762 -- 2
```

   c. Extend your Perl script with code that computes and prints out the average runtime by dividing `$runtime` by `$count`. The output should look as follows:

```
Average runtime: 9.881
```

4. Let us try to devise Perl code that finds and counts pairs of `Generating–Sorting` in `$_`.

   a. Add the following code at the end of your Perl script, save the file and then execute it.

```
$count = 0;
$_ = $log1;
while (/Generating.*Sorting/g) {
  $count++;
  print "3: Match $count found at $-[0] to ",$+[0]-1,"\n";
}
print "Total number of matches found: $count\n\n";
```

   Disappointingly the output will be

```
Total number of matches found: 0
```

   b. Why does the code not work as expected?

   c. In a first step try to correct the code by adding a modifier to `/ /` so that at least one match is found.

   d. Obviously we should really find two matches. Change the regular expression so that two matches are found and the output is

```
3: Match 1 found at 0 to 54
3: Match 2 found at 77 to 132
Total number of matches found: 2
```

   e. Change the regular expression and the modifier again, this time in such a way that `Generating` and `Sorting` both have to occur at the start of a line in order to be matched. The modified code should only find one match.

5. The general format of a URL is

   `scheme://domain:port/path?query_string#fragment_id`

   In the following we assume that a URL only contains ASCII characters. Furthermore, we assume that

   - `scheme` consists of a sequence of characters beginning with a letter and followed by any combination of letters (`a`–`z`, `A`–`Z`), digits (`0`–`9`), plus (`+`), period (`.`), or hyphen (`-`);
   - `domain` is a sequence of *labels* concatenated by a period (`.`), each label is a sequence of letters, digits and hyphen that does not start nor end with a hyphen;
   - `port` is a natural number, `:port` is optional;
   - `path` does not contain `?` or `#`, `/path` is optional;
   - `query_string` does not contain `#`, `?query_string` is optional;
   - `fragment_id` can contain arbitrary characters, `#fragment_id` is optional.

   These are simplifying assumptions, for a full definition of the syntax of URLs and URIs see https://tools.ietf.org/html/rfc3986 (in your own time).

   a. Add the following code at the end of your Perl script.

```
@urls = (
 "http://www.example.com",
 "http://www80.local.com:80/",
 "ftp://www.example.com/here/to/information.html",
 "HTTP://www.example.com/some/more/information.html#item",
```

```
 "HTTP://www.example.com/some/../info.html#item",
 "https://www.example.com:80/here/there/../search?query",
 "web+://www.1--2.com/some/more/perl?query+me+this",
 "https://www.ex221.ac.uk:442/perl/rulez?all+q#all.time");
foreach (@urls) {
  print "URL: $_\n";
  ($scheme,$domain,$port,$path,$query,$fragment) =
          (/(.)(.)(.)(.)(.)(.)/);
  print "SCHEME: $scheme, DOMAIN: $domain, PORT: $port\n";
  print "PATH: $path\n"; print "QUERY: $query\n";
  print "FRAGMENT: $fragment\n\n";
}
```

b. Now change the regular expression in the code above so that it correctly separates the five components of a URL and use the sample URLs to test that it works as expected.
To find the right regular expression it might be useful to experiment with
http://www.perlfect.com/articles/regextutor.shtml

c. Add code that removes dot-segments from `$path`.
Hint: Refer to the lecture notes for the substitution that is required to do so.

6. Repeat some of the code examples that we have seen in the last two lectures.

a. Add the following code to your Perl script, save it, and execute it.

```
$_ = "ab 11 cd 22 ef 33";
if (/\d+/g)  { print "1: Match starts at $-[0]: $&\n" }
if (/[a-z]+/g) { print "2: Match starts at $-[0]: $&\n" }
if (/\d+/g)  { print "3: Match starts at $-[0]: $&\n" }
```

b. Check that the output is as expected.

c. Add the following code to your Perl script, save it, and execute it.

```
$_ = "ab 11 cd 22 ef 33";
if (/\d+/g) { print "4: Match starts at $-[0]: $&\n" }
if (/ab/g) { print "5: Match starts at $-[0]: $&\n" }
if (/\d+/g) { print "6: Match starts at $-[0]: $&\n" }
```

d. Check that the output is as expected.

e. Correct the code in 6c above so that the output is

```
4: Match starts at 3: 11
6: Match starts at 9: 22
```

f. Add the following code to your Perl script, save it, and execute it.

```
$_ = "Bart teases Lisa";
@keywords = ("bart","lisa","marge","L\w+","t\\w+");
while ($keyword = shift(@keywords)) {
  print "Match found for $keyword: $&\n" if /$keyword/i;
}
```

g. Check that the output is

```
Match found for bart: Bart
Match found for lisa: Lisa
Match found for t\w+: teases
```

Why is there no match for "L\w+"?

h. Add the following code to your Perl script, fill in your name, save it, and execute it.

```
$name = "Dr Ullrich Hustadt";
$name =~ s/(Mr|Ms|Mrs|Dr)?\s*(\w+)\s+(\w+)/\U$3\E, $2/;
print "$name\n";

$name = "<fill in your name>";
$name =~ s/(Mr|Ms|Mrs|Dr)?\s*(\w+)\s+(\w+)/\U$3\E, $2/;
print "$name\n";
```

i. Add the following code to your Perl script, save it, and execute it.

```
$text = "The temperature is 105 degrees Fahrenheit";
$text =~ s!(\d+) degrees Fahrenheit!
          (($1-32)*5/9)." degrees Celsius"!e;
print "$text\n";
$text =~ s!(\d+\.\d+)!sprintf("%d",$1+0.5)!e;
print "$text\n";
```

j. Simplify the code in Exercise 6i above so that the transformation from degrees Fahrenheit to degrees Celcius is done is a single step.

7. The following exercise is about Perl subroutines.

a. Read

http://perldoc.perl.org/perlsub.html

and the lecture notes to get an understanding of Perl subroutines.

b. Open a text editor and enter the following Perl code:

```
#!/usr/bin/perl
# Author: <your name>
# Practical 2: Perl Subroutines [perl02B]

sub sum {
  return $_[0] + $_[1];
}

$first = "3";
$second = 4;
$result = sum($first,$second);
print "The sum of $first and $second is $result\n";
```

Replace <your name> with your own name.

c. Save the code to a file named perl02B in some appropriate directory, adjust its access rights as shown in Exercise 1d, then execute the file.

d. Refine the subroutine sum so that it can compute the sum of any list of numbers.
Hint: The answer is in the lecture notes.

8. The subroutine sum in Exercise 7 returns a single scalar value. Returning a list is just as simple, as the following exercise shows.

a. Add the following code to your Perl script, save it, and execute it.

```
sub format_names {
   my @names=@_;
   foreach (@names) {
      s/(Mr|Ms|Mrs|Dr)?\s*(\w+)\s+(\w+)/\U$3\E, $2/
   }
   return @names;
}

@names = ("Dr Martin Gairing", "Dr Othon Michail");
@output = format_names(@names);
print "Input: ",join("; ",@names), "\n";
print "Output: ",join("; ",@output), "\n";
```

b. Check that the output is

```
Input: Dr Martin Gairing, Dr Othon Michail
Output: GAIRING, Martin; MICHAIL, Othon
```

c. Note that we have used `format_names` instead of `&format_names` in the call of the subroutine. Note also that there are two arrays called `names`, one inside the subrouting `format_names` and one outside. While the latter is *global*, the former is *local* to `format_names`, due to the use of the `my` operator when it was first used. The two arrays do not interact as the output shows.

9. Perl also allows nested subroutines but these have rather peculiar behaviour.

a. Add the following code to your Perl script, save it, and execute it.

```
sub outer_sub {
   my $var = 10;
   sub inner_sub { print "$var\n"; }
   inner_sub();
   $var+=10;
}
print "1st call of outer_sub: "; outer_sub();
print "value of \$var:     ", $var, "\n";
print "1st call of inner_sub: "; inner_sub();
print "2nd call of outer_sub: "; outer_sub();
```

b. Check that the output is

```
1st call of outer_sub: 10
value of $var:
1st call of inner_sub: 20
2nd call of outer_sub: 20
```

Note that it is possible to call `inner_sub` outside the scope of `outer_sub`. So, Perl does not actually 'hide' a subroutine that is defined inside another subroutine.

However, that raises the question which `$var` a call of `inner_sub` is supposed to print. The variable `$var` is local to `outer_sub`, it is not accessible from outside `outer_sub`, as shown by the output above. Also, remember that every call of `outer_sub` creates a new instance of `$var`.

It turns out that any call of `inner_sub` refers to the very first instance of `$var`, that is, the one created the first time `outer_sub` was called.

This explains the output `1st call of inner_sub: 20`: The value 20 was the value of `$var` on completion of the first call of `outer_sub`.

This is also the reason for the output `2nd call of outer_sub: 20`. While the second call of `outer_sub` creates a new instance of `$var`, `inner_sub` still refers to the first instance of `$var` created by the first call of `outer_sub`.