Experiment 7

8th April 2019

Aim: To be able to write Perl scripts.

----

1. Create a text file and answer the following queries :

a) Search for the pattern ‘apple’ in the file and display the number of occurences.

b) Count the number of words that ends with ‘e’.

c) Count the number of words that starts with ‘ap’.

d) Search for words containing ‘a’ or ‘s’.

e) Search for words containing zero or more occurrence of ‘e’.

f) Search for words containing one or more occurrence of ‘e’.

g) Search for words containing the letters ‘l’ and ‘m’, with any number of characters in between.

Preface:

Following table lists the regular expression syntax that is available in Python.

Sr. No.	Pattern & Description
1	^ Matches beginning of line.
2	**$** Matches end of line.
3	**.** Matches any single character except newline. Using m option allows it to match newline as well.
4	**[...]** Matches any single character in brackets.
5	**[^...]** Matches any single character not in brackets.
6	*** Matches 0 or more occurrences of preceding expression.
7	+ Matches 1 or more occurrence of preceding expression.
8	**?** Matches 0 or 1 occurrence of preceding expression.

9	**{ n}** Matches exactly n number of occurrences of preceding expression.
10	**{ n,}** Matches n or more occurrences of preceding expression.
11	**{ n, m}** Matches at least n and at most m occurrences of preceding expression.
12	**a| b** Matches either a or b.
13	**\w** Matches word characters.
14	**\W** Matches nonword characters.
15	**\s** Matches whitespace. Equivalent to [\t\n\r\f].
16	**\S** Matches nonwhitespace.
17	**\d**

	Matches digits. Equivalent to [0-9].
18	**\D** Matches nondigits.
19	**\A** Matches beginning of string.
20	**\Z** Matches end of string. If a newline exists, it matches just before newline.
21	**\z** Matches end of string.
22	**\G** Matches point where last match finished.
23	**\b** Matches word boundaries when outside brackets. Matches backspace (0x08) when inside brackets.
24	**\B** Matches nonword boundaries.
25	**\n, \t, etc.** Matches newlines, carriage returns, tabs, etc.

26	**\1...\9** Matches nth grouped subexpression.
27	**\10** Matches nth grouped subexpression if it matched already. Otherwise refers to the octal representation of a character code.
28	**[aeiou]** Matches a single character in the given set
29	**[^aeiou]** Matches a single character outside the given set

The techniques used in the above table is used to write the corresponding 'regex' that forms a necessary part of the following Perl scripts.

The basic method for applying a regular expression is to use the pattern binding operators =~ and **!**~. The first operator is a test and assignment operator.

There are three regular expression operators within Perl.

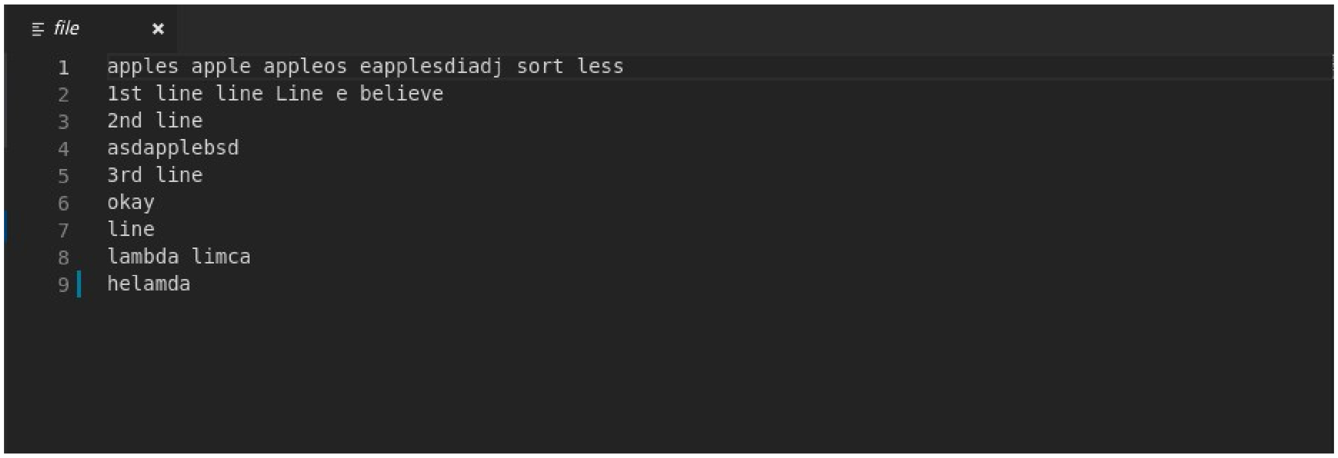
1. Match Regular Expression - m//
2. Substitute Regular Expression - s///
3. Transliterate Regular Expression – tr///

The forward slashes in each case act as delimiters for the regular expression (regex) that you are specifying. If you are comfortable with any other delimiter, then you can use in place of forward slash.

Solutions:

NOTE:

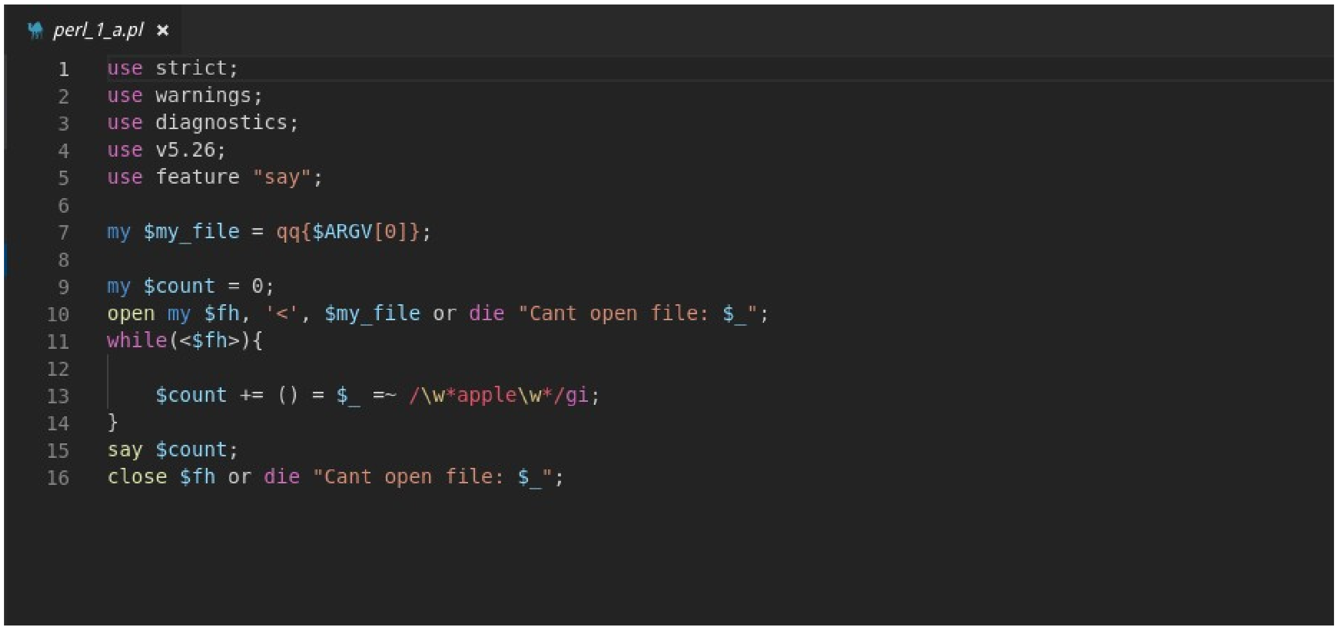
The file used in all the following example is given below.



```
apples apple appleos eapplesdiadj sort less
1st line line Line e believe
2nd line
asdapplebsd
3rd line
okay
line
lambda limca
helamda
```

1. a)

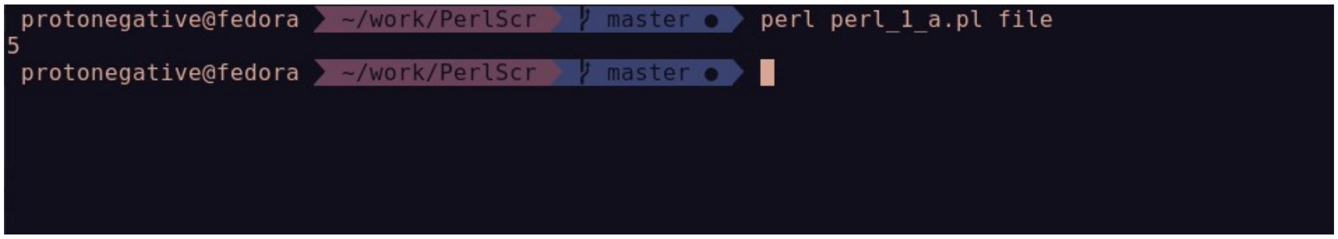
The code is as follows:



```
use strict;
use warnings;
use diagnostics;
use v5.26;
use feature "say";

my $my_file = qq{$ARGV[0]};

my $count = 0;
open my $fh, '<', $my_file or die "Cant open file: $_";
while(<$fh>){

    $count += () = $_ =~ /\w*apple\w*/gi;
}
say $count;
close $fh or die "Cant open file: $_";
```

Here the regex /\w*apple\w*/gi is used to find all sub-strings apple in the file.

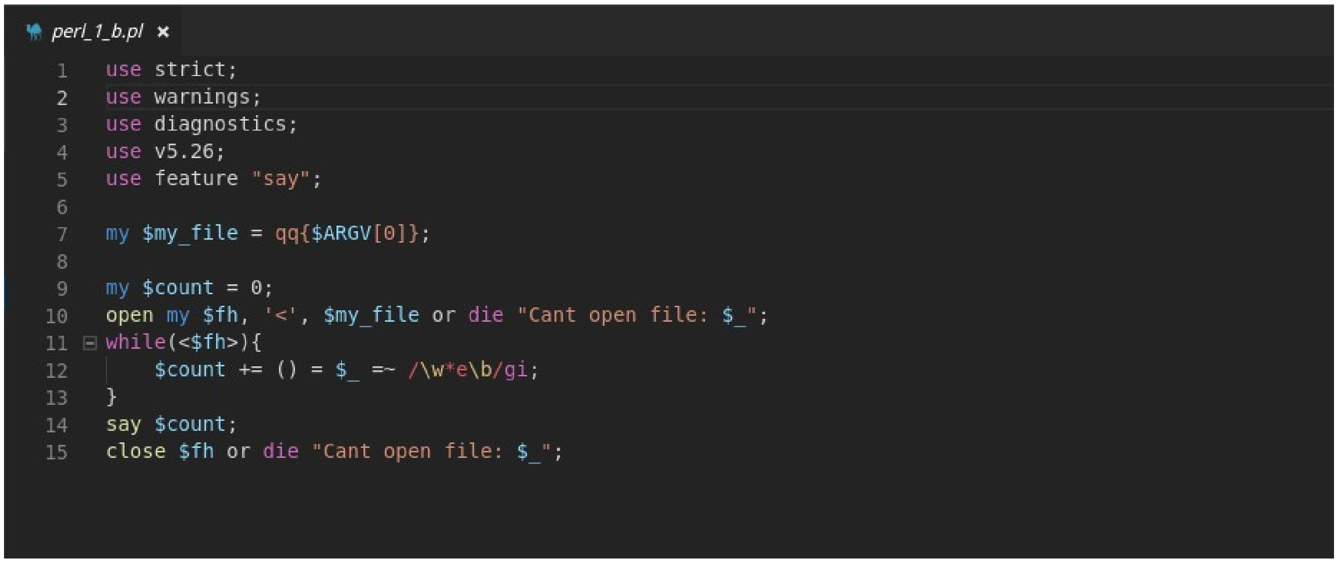
The output of the above code on ‘file’ is:



```
 protonegative@fedora  ~/work/PerlScr  master ●  perl perl_1_a.pl file
5
 protonegative@fedora  ~/work/PerlScr  master ● 
```

1. b)

The code is as follows:

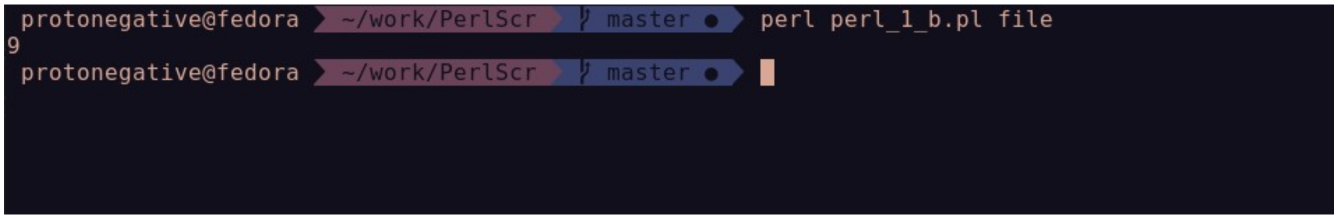


perl_1_b.pl

```perl
use strict;
use warnings;
use diagnostics;
use v5.26;
use feature "say";

my $my_file = qq{$ARGV[0]};

my $count = 0;
open my $fh, '<', $my_file or die "Cant open file: $_";
while(<$fh>){
    $count += () = $_ =~ /\w*e\b/gi;
}
say $count;
close $fh or die "Cant open file: $_";
```

Here the regex /\w*e\b/gi is used to find all words that end with e.

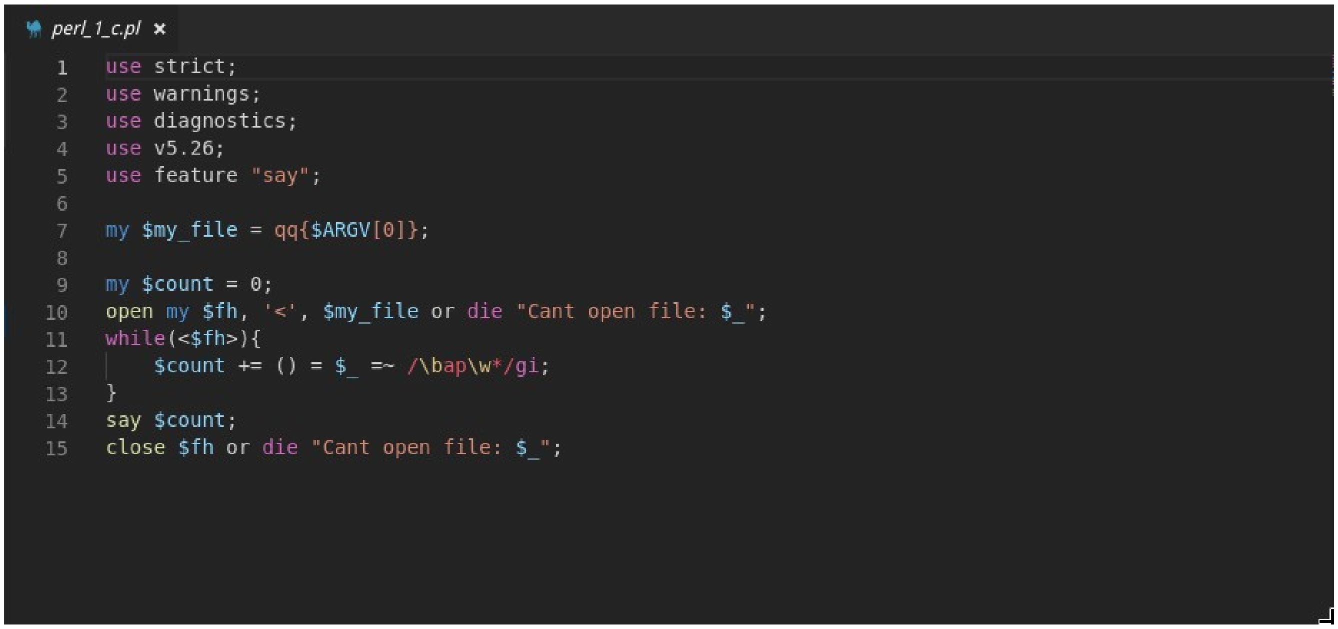
The output of the above code on ‘file’ is:



```
 protonegative@fedora  ~/work/PerlScr  master ●  perl perl_1_b.pl file
9
 protonegative@fedora  ~/work/PerlScr  master ● 
```

1. c)

The code is as follows:

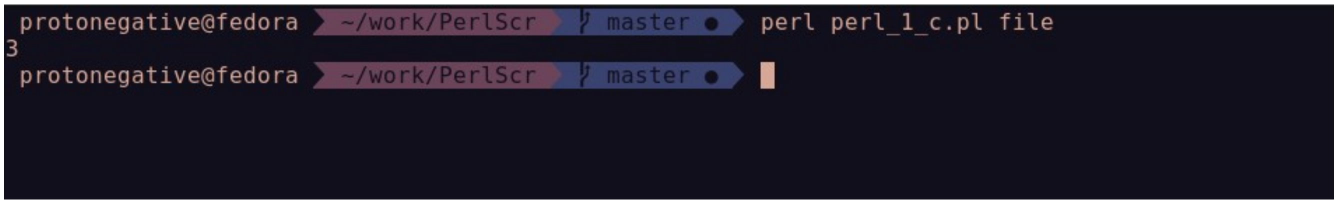


perl_1_c.pl ×

```perl
use strict;
use warnings;
use diagnostics;
use v5.26;
use feature "say";

my $my_file = qq{$ARGV[0]};

my $count = 0;
open my $fh, '<', $my_file or die "Cant open file: $_";
while(<$fh>){
    $count += () = $_ =~ /\bap\w*/gi;
}
say $count;
close $fh or die "Cant open file: $_";
```

Here the regex /\bap\w*/gi is used to find all words that start with ap.

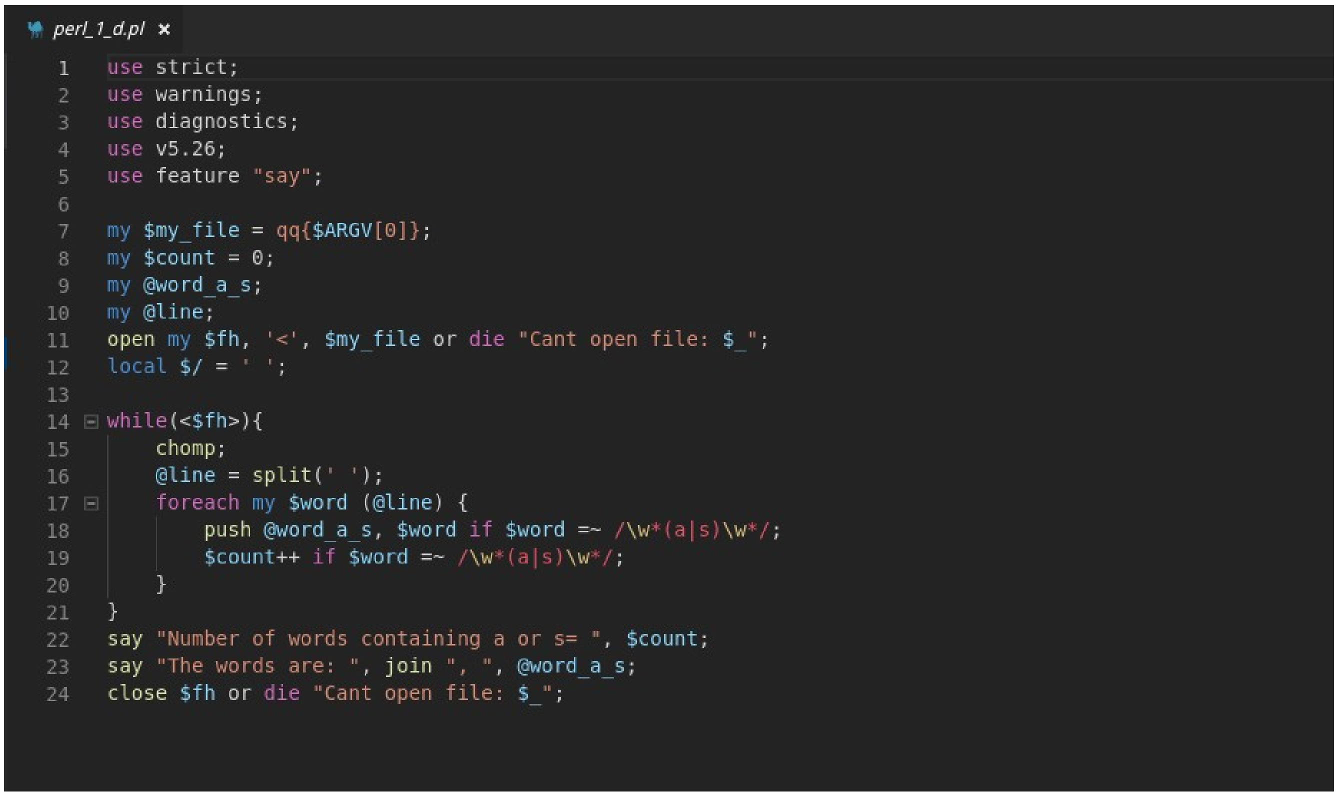
The output of the above code on ‘file’ is:



```
 protonegative@fedora  ~/work/PerlScr  master ●  perl perl_1_c.pl file
3
 protonegative@fedora  ~/work/PerlScr  master ● 
```

1. d)

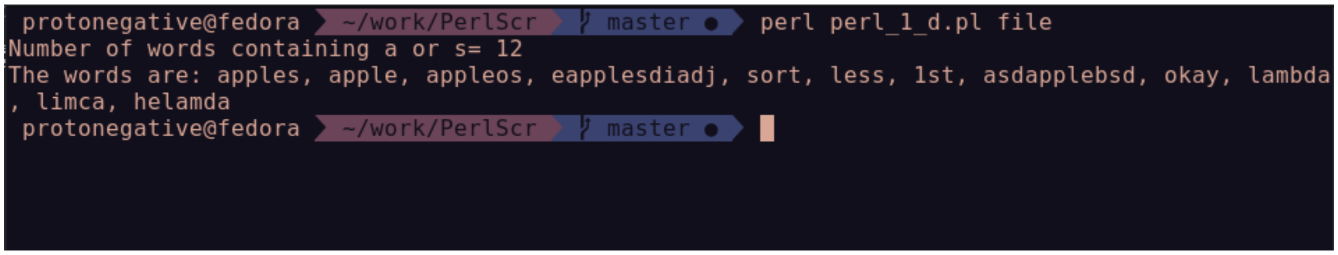
The code is as follows:



```perl
use strict;
use warnings;
use diagnostics;
use v5.26;
use feature "say";

my $my_file = qq{$ARGV[0]};
my $count = 0;
my @word_a_s;
my @line;
open my $fh, '<', $my_file or die "Cant open file: $_";
local $/ = ' ';

while(<$fh>){
    chomp;
    @line = split(' ');
    foreach my $word (@line) {
        push @word_a_s, $word if $word =~ /\w*(a|s)\w*/;
        $count++ if $word =~ /\w*(a|s)\w*/;
    }
}
say "Number of words containing a or s= ", $count;
say "The words are: ", join ", ", @word_a_s;
close $fh or die "Cant open file: $_";
```

Here the regex /\w*(a|s)\w*/gi is used to find all words that has a or s.

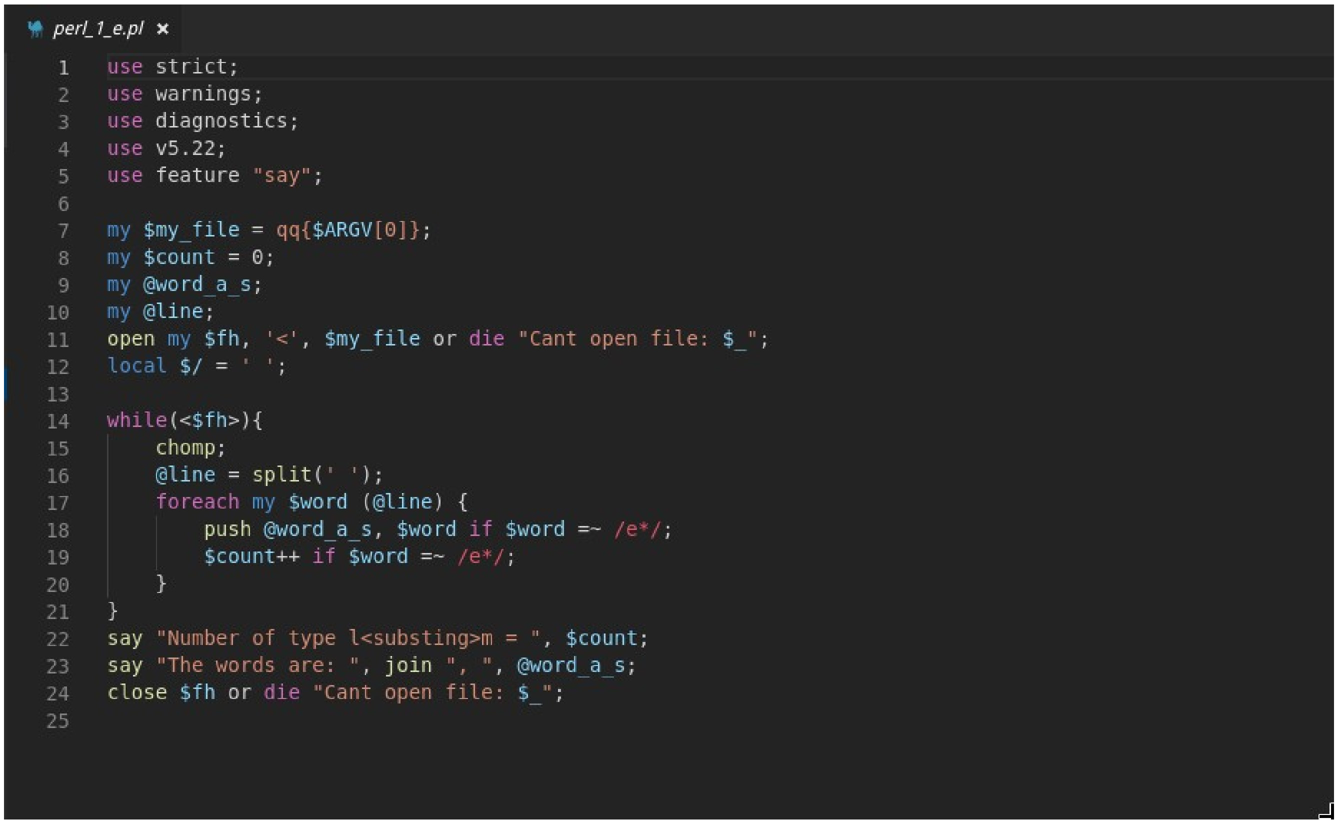
The output of the above code on ‘file’ is:



```
 protonegative@fedora  ~/work/PerlScr   master ●  perl perl_1_d.pl file
Number of words containing a or s= 12
The words are: apples, apple, appleos, eapplesdiadj, sort, less, 1st, asdapplebsd, okay, lambda
, limca, helamda
 protonegative@fedora  ~/work/PerlScr   master ● 
```

1. e)

The code is as follows:

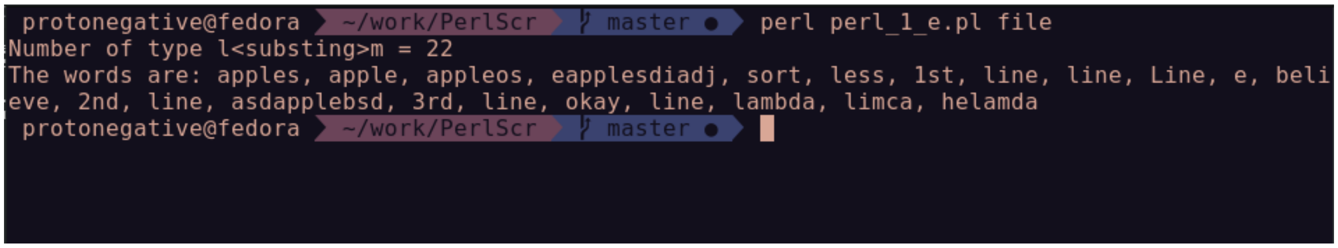


perl_1_e.pl ×

```perl
use strict;
use warnings;
use diagnostics;
use v5.22;
use feature "say";

my $my_file = qq{$ARGV[0]};
my $count = 0;
my @word_a_s;
my @line;
open my $fh, '<', $my_file or die "Cant open file: $_";
local $/ = ' ';

while(<$fh>){
    chomp;
    @line = split(' ');
    foreach my $word (@line) {
        push @word_a_s, $word if $word =~ /e*/;
        $count++ if $word =~ /e*/;
    }
}
say "Number of type l<substing>m = ", $count;
say "The words are: ", join ", ", @word_a_s;
close $fh or die "Cant open file: $_";

```

Here the regex /e*/gi is used to find zero or more occurrences of e .

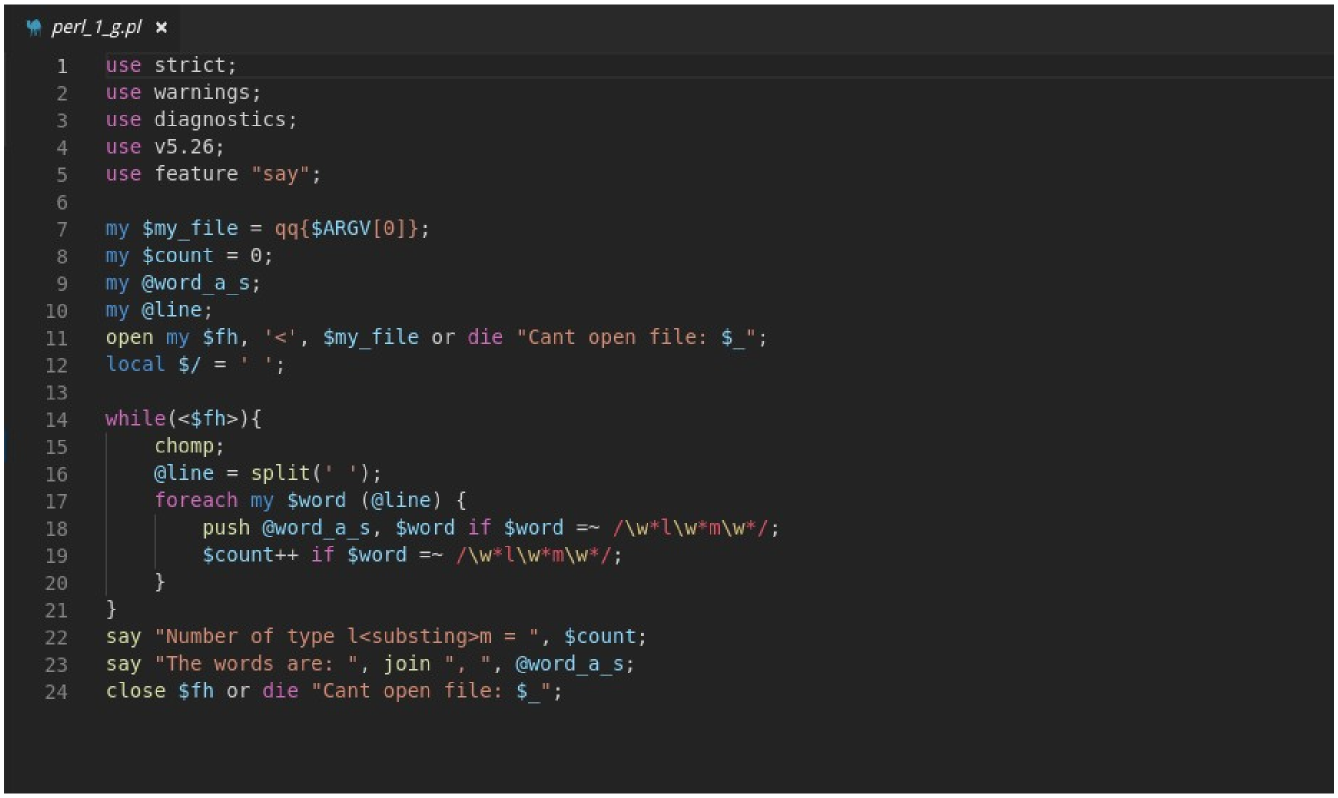
The output of the above code on ‘file’ is:



```
 protonegative@fedora  ~/work/PerlScr  ᚴ master ●  perl perl_1_e.pl file
Number of type l<substing>m = 22
The words are: apples, apple, appleos, eapplesdiadj, sort, less, 1st, line, line, Line, e, beli
eve, 2nd, line, asdapplebsd, 3rd, line, okay, line, lambda, limca, helamda
 protonegative@fedora  ~/work/PerlScr  ᚴ master ● 
```

1. g)

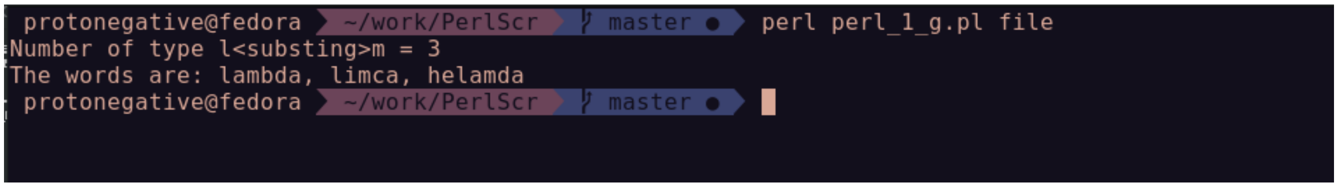
The code is as follows:



```
perl_1_g.pl
use strict;
use warnings;
use diagnostics;
use v5.26;
use feature "say";

my $my_file = qq{$ARGV[0]};
my $count = 0;
my @word_a_s;
my @line;
open my $fh, '<', $my_file or die "Cant open file: $_";
local $/ = ' ';

while(<$fh>){
    chomp;
    @line = split(' ');
    foreach my $word (@line) {
        push @word_a_s, $word if $word =~ /\w*l\w*m\w*/;
        $count++ if $word =~ /\w*l\w*m\w*/;
    }
}
say "Number of type l<substing>m = ", $count;
say "The words are: ", join ", ", @word_a_s;
close $fh or die "Cant open file: $_";
```

Here the regex /\w*l\w*m\w*/gi is used to find words containing l and m with any sub-string in between .

The output of the above code on ‘file’ is:



```
 protonegative@fedora  ~/work/PerlScr  master ●  perl perl_1_g.pl file
Number of type l<substing>m = 3
The words are: lambda, limca, helamda
 protonegative@fedora  ~/work/PerlScr  master ● 
```

Result: Learned to write Perl scripts.

----

eof
