Exercises - Part 1

2.1 Write a program that computes the circumference of a circle with a radius of 12.5. The circumference is \(2\pi\) times the radius (and \(\pi\) approximates 3.141592654).

2.2 Modify the program from the previous exercise to prompt for, and accept, a radius from the person running the program.

2.3 Write a program to take in two numbers that prints out the result of the two numbers multiplied together.

2.4 Write a program that reads in a string and a number, and then prints out the string the number of times requested. (Hint: use the ‘\(\times\)’ operator)

3.1 Write a program that reads a list of strings, and prints out the list in reverse order.

3.2 Write a program that reads in a number and a series of lines, then prints one of the lines from the list, as selected by the number.

3.3 Write a program that reads in a list of strings, then prints one chosen at random.

To select a random number within your array, you need to do two things:
- put \(\text{srand;}\) near the top of your program to initialise the Random Number Generator
- use \(\text{rand}(@\text{my\_array});\) to generate a random number between 0 and the last element in the array
4.1 Write a program that asks for the temperature outside (us oldies work in Farenheit). The program should print **too hot** if the temperature is above 72, and **too cold** otherwise.

4.2 Modify the program from the previous exercise so that it prints **too hot** if the temperature is above 75, **too cold** if it’s below 68, and **just right** if it’s between 68 & 75.

4.3 Write a program that reads in a list of numbers (one per line), until the number **999** is entered, then it prints the sum of all the numbers entered. Be sure not to add the 999.
For example, if the numbers 1, 2, 3, 999 are entered, the answer is 6 (1+2+3).

4.4 Write a program that reads in a list of strings (on separate lines), then prints out the list in reverse order. Do this *without* using the reverse operator.

4.5 Write a program that prints a table of numbers and their squares, from 0 to 32.

eg:

```
0 -> 0
1 -> 1
2 -> 4
3 -> 9
```

... etc

Try to come up with a way where you don’t have 0-32 in a list, and one where you do.

For nicely formatted output, you can use:

```
printf "\%5g -> \%8g\n", $a, $b;
```

which will print $a as a 5-column number, $b as an 8-column number.
Exercises - Part 2

5.1 Write a program that reads and prints a string, and its mapped value, according to the mapping presented below.

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>Apple</td>
</tr>
<tr>
<td>Green</td>
<td>Leaves</td>
</tr>
<tr>
<td>Blue</td>
<td>Ocean</td>
</tr>
</tbody>
</table>

5.2 Write a program that reads a series of words (with one word per line) until End-Of-File, and then prints a summary of how many times each word was seen. (For an extra challenge, sort the words into ascending ASCII order in the output).

6.1 Write a program that acts like cat, but reverses the order of the lines.

6.2 Write a program that reads a list of strings, and prints the strings in a right justified, 20 character column. For example, inputting hello (new line) goodbye (new line) prints hello on one row, and goodbye on the next row, both aligned such that the o and e are in the same (19th) column.

6.3 Modify the program from the previous exercise to allow the user to enter the column width. For example, entering the lines 20, hello, goodbye produces the same output as above, however entering the lines 30, hello, goodbye prints the two words in a column that is justified 30 places.
7.1 Construct regular expressions that match:
1) at least one 'a', followed by any number of 'b's
2) any number of back-slashes, followed by any number of stars
3) three consecutive copies of whatever is contained within the variable $whatever
4) any five characters, including the newline character
5) the same word written two or more times in a row, where “word” is defined as a non-empty sequence of non-whitespace characters. (You will need to look this one up - it's not in the slides!)

7.2 (a) Write a program that accepts a list of words on STDIN and looks for a line containing all five vowels (specifically a,e,i,o,u). Run this program on /usr/dict/words.
   ie, run "myprog < /usr/dict/words"

(b) Modify the program so that the vowels have to be in order.

(c) For an extra challenge, make a further modification such that all the vowels must be present, and in ascending order - ie, no “e” before an “a”, no “i” before an “e”, and so on.

7.3 Write a program that looks through /home/kiz/test.passwd.file (on STDIN), printing the login name and real name of each user.
   Hint: use “split” to break the lines up into fields, then “s///” to get rid of the parts of the comment field after the first comma

7.4 Write a program that looks through /home/kiz/test.passwd.file (on STDIN), for users with the same first name, and prints out those names
   Hint: after splitting the line, create a hash with the names as the keys and the number of times it was seen for the value. Now you can look for the hash entries with a value greater than 1.

7.5 Repeat the last exercise, however report the login IDs for each person.
   Hint: instead of storing a count in the hash, store a list of names, separated by spaces. When finished, look for those entries that contain at least one space.