ECMM171: Programming for Engineers

Lab 2: lists and loops

The purpose of this workshop is to allow you to try out the programming aspects you have learned about in todays lectures. In this lab, you will:

- investigate **for** and **while** loops.
- create and manipulate lists, slice lists, and construct list comprehensions.

---

1 Questions

1. Without using any of the built-in string functions, write a program that uses a **for** loop to traverse the sentence:

   "Homer was a happy hamster who lived in a heart shaped house. Homer had the best cage, it was almost as big as a doghouse."

   and tells you how many occurrences of the letter 'h' were in your text (ignore the capital 'H' for the number of occurrences). Test your answer is correct by using the `.count()` function we saw in the lectures.

2. Starting from the `user_input.py` example from the lecture notes, write a program that:
   - reads numbers from a user into a list until the user enters the character 'n';
   - processes the list, using a `for` loop, to print out the maximum and minimum number that the user entered – without using the Python `min` and `max` function!

3. Using either a **for** or **while** loop, write a program to test if a user-input value lies in a list. Your code should start with something like the following:

   ```
   my_list = [1, 6, 4, 7, 4, 5, 7]
   user_input = int(input("Search for a number: "))
   # your code here that prints whether `user_input` lies in `my_list`.
   ```

   Don't cheat by using Python's **in** keyword!

4. Write a list comprehension that removes words from a list if they contain the letter 'n'. For example:

   ```
   >>> words = [ 'I', 'like', 'pythons', 'and', 'rabbits' ]
   >>> b = [ ... for word in a if ... ]
   >>> print(b)
   [ 'I', 'like', 'rabbits' ]
   ```

5. Consider the list `a = [1, 2, 3, 4, 5, 6, 7, 8]`. Using *only* the slicing notation we discussed for lists in the lectures, work out what slices you need to select to get the following output:

```
>>> a = [1, 2, 3, 4, 5, 6, 7, 8]
>>> a[...]  # part (a)
[3, 4, 5, 6]
>>> a[...]  # part (b)
[2, 4, 6, 8]
>>> a[...]  # part (c)
[4, 3, 2, 1]
>>> a[...]  # part (d)
[8, 6, 4, 2]
>>> a[...]  # part (e)
[7, 5, 3]
```

6. You might remember that we can approximate functions using their Taylor series, e.g.

$$\frac{1}{1-x} \approx \sum_{k=1}^{n} x^k = 1 + x + x^2 + x^3 + \cdots + x^n$$

This only works if $-1 < x < 1$. Write a program that:

(a) reads values of x from the user (and checks they are valid).

(b) calculates the approximation of $1/(1-x)$ to within a value of 10^{-6} of the exact answer.

(c) prints out how many terms n were required to calculate the approximation.