1. Write and test a function that takes a string as a parameter and returns a sorted list of all the unique letters used in the string. So, if the string is cheese, the list returned should be ['c', 'e', 'h', 's'].

Code:

```
def sorted_unique_letters(s):
   return sorted(set(s))
items = ['c', 'h', 'e', 'e', 's','e']
print(sorted_unique_letters(items))
```

Output:

```
['c', 'e', 'h', 's']
```

2. Write and test three functions that each take two words (strings) as parameters and return sorted lists (as defined above) representing respectively:

Letters that appear in at least one of the two words.

Letters that appear in both words.

Letters that appear in either word, but not in both.

Hint: These could all be done programmatically, but consider carefully what topic we have been discussing this week! Each function can be exactly one line.

Code:

```
def union_letters(word1, word2):
   return sorted(set(word1) | set(word2))  # Letters that appear in at least one word

def intersection_letters(word1, word2):
   return sorted(set(word1) & set(word2))  # Letters that appear in both words

def difference_letters(word1, word2):
   return sorted(set(word1) ^ set(word2))  # Letters that appear in either word, but not both

# User input
word1 = input("Enter first word: ")
word2 = input("Enter second word: ")

print("Union:", union_letters(word1, word2))
print("Intersection:", intersection_letters(word1, word2))
print("Difference:", difference_letters(word1, word2))
```

Output:

```
Enter first word:  Apeal
Enter second word:  Babu
Union: ['A', 'B', 'a', 'b', 'e', 'l', 'p', 'u']
Intersection: ['a']
Difference: ['A', 'B', 'b', 'e', 'l', 'p', 'u']
```

3. Write a program that manages a list of countries and their capital cities. It should prompt the user to enter the name of a country. If the program already "knows" the name of the capital city, it should display it. Otherwise it should ask the user to enter it. This should carry on until the user terminates the program (how this happens is up to you).

Note: A good solution to this task will be able to cope with the country being entered variously as, for example, "Wales", "wales", "WALES" and so on.

Code:

```
def manage_capitals():
  country_capitals = {}

  while True:
    country = input("Enter a country (or type 'exit' to quit): ").strip()

    if country.lower() == 'exit':
      print("Exiting the program...")
      break

    country_lower = country.lower()

    if country_lower in country_capitals:
      print(f"The capital city of {country} is {country_capitals[country_lower]}.")
    else:
      capital = input(f"I don't know the capital of {country}. Please enter it: ").strip()
      country_capitals[country_lower] = capital
      print(f"Thank you! I've stored the capital of {country} as {capital}.")

manage_capitals()
```

Output:

```
Enter a country (or type 'exit' to quit):  Nepal
I don't know the capital of Nepal. Please enter it:  Kathmandu
Thank you! I've stored the capital of Nepal as Kathmandu.
Enter a country (or type 'exit' to quit):  exit
Exiting the program...
```

4. One approach to analysing some encrypted data where a substitution is suspected is frequency analysis. A count of the different symbols in the message can be used to identify the language used, and sometimes some of the letters. In English, the most common letter is "e", and so the symbol representing "e" should appear most in the encrypted text. Write a program that processes a string representing a message and reports the six most common letters, along with the number of times they appear. Case should not matter, so "E" and "e" are considered the same. Hint: There are many ways to do this. It is obviously a dictionary, but we will want zero counts, so some initialisation is needed. Also, sorting dictionaries is tricky, so best to ignore that initially, and then check the usual resources for the runes.

Code:

```
from collections import defaultdict

def frequency_analysis(message):
    letter_counts = defaultdict(int)

    message = message.lower()

    for char in message:
        if char.isalpha():
            letter_counts[char] += 1

    sorted_counts = sorted(letter_counts.items(), key=lambda x: x[1], reverse=True)

    print("Most common letters and their counts:")
    for letter, count in sorted_counts[:6]:
        print(f"{letter}: {count}")

message = input("Enter a message: ")
frequency_analysis(message)
```

Output:

```
Enter a message:  Apeal Babu Rokaya
Most common letters and their counts:
a: 5
b: 2
p: 1
e: 1
l: 1
u: 1
```