Program 7

January 16, 2025

[3]:

```python
'''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'].'''

def unique_sorted_letters(word):
    unique_letters = set()

    for letter in word:
        unique_letters.add(letter)

    sorted_unique_letters=sorted(list(unique_letters))
    return sorted_unique_letters

def main():
    user_input = input("Enter a word: ")
    result = unique_sorted_letters(user_input)
    print(f"Unique sorted letters: {result}")

main()
```

```
Enter a word:  jkahsd kahdjhsahdhas
```

```
Unique sorted letters: [' ', 'a', 'd', 'h', 'j', 'k', 's']
```

[4]:

```python
'''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.'''

def unique_sorted_union(word1, word2):
    # Union of the sets
    union= sorted(list(set(word1) | set(word2)))
    return union

def unique_sorted_intersection(word1, word2):
    # Intersection of the sets
    intersection= sorted(list(set(word1) & set(word2)))
    return intersection

def unique_sorted_difference(word1, word2):
    # Symmetric difference of the sets
    difference= sorted(list(set(word1) ^ set(word2)))
    return difference

def main():
    word1 = "Gorgeous"
    word2 = "IamGroot"

    print(f"Letters that appear in at least one of the words:␣
  ↪{unique_sorted_union(word1, word2)}")
    print(f"Letters that appear in both words:␣
  ↪{unique_sorted_intersection(word1, word2)}")
    print(f"Letters that appear in either word but not both:␣
  ↪{unique_sorted_difference(word1, word2)}")

main()
```

```
Letters that appear in at least one of the words: ['G', 'I', 'a', 'e', 'g', 'm',
'o', 'r', 's', 't', 'u']
Letters that appear in both words: ['G', 'o', 'r']
Letters that appear in either word but not both: ['I', 'a', 'e', 'g', 'm', 's',
't', 'u']
```

[5]:

```
'''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.'''

def main():
    countries = {
        "nepal": "kathmandu",
        "france": "paris",
        "japan": "tokyo"
    }

    while True:
        country = input("Enter the name of a country (You can type 'exit' to␣
 ↪quit): ").lower()

        if country == "exit":
            print("Exiting the program. Goodbye!")
            break

        if country in countries:
            print(f"The capital city of {country.capitalize()} is␣
 ↪{countries[country].capitalize()}.")
        else:
            capital = input(f"I don't know the capital of {country.
 ↪capitalize()}. Please enter it: ").strip().lower()
            countries[country] = capital
            print(f"Added {country.capitalize()} and its capital {capital.
 ↪capitalize()} to the list.")


main()
```

```
Enter the name of a country (or type 'exit' to quit):  Nepal

The capital city of Nepal is Kathmandu.

Enter the name of a country (or type 'exit' to quit):  exit

Exiting the program. Goodbye!
```

[7]:

```
'''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.'''

from collections import Counter

def frequency_analysis(message):
    normalized_message = [char.lower() for char in message if char.isalpha()]
    letter_counts = Counter(normalized_message)
    most_common = letter_counts.most_common(6)

    return most_common

def main():
    message = input("Enter the encrypted message: ")
    most_common_letters = frequency_analysis(message)

    print("\nSix most common letters and their frequencies:")
    for letter, count in most_common_letters:
        print(f"Letter: {letter}, Count: {count}")

main()
```

```
Enter the encrypted message:  Nikhil


Six most common letters and their frequencies:
Letter: i, Count: 2
Letter: n, Count: 1
Letter: k, Count: 1
Letter: h, Count: 1
Letter: l, Count: 1
```

```
[ ]:
```