program7

January 3, 2025

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'].

[2]:
```
def sort_string(string):
    unique_letters = set(char.lower() for char in string if char.isalpha())
    return sorted(unique_letters)

print(sort_string("Hello, Goodmorning!"))
print(sort_string("78294"))
```

```
['d', 'e', 'g', 'h', 'i', 'l', 'm', 'n', 'o', 'r']
[]
```

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.

[10]:
```
def sort_string(string):
    unique_letters = set(char.lower() for char in string if char.isalpha())
    return sorted(unique_letters)

atleast_one_letter= lambda word1,word2: sorted(set(sort_string(word1)) |␣
 ↪set(sort_string(word2)))

letter_both_word = lambda word1, word2: sorted(set(sort_string(word1)) &␣
 ↪set(sort_string(word2)))


letter_either_not_both = lambda word1, word2: sorted(set(sort_string(word1)) ^␣
 ↪set(sort_string(word2)))

print(atleast_one_letter("apple", "mango"))
print(letter_both_word("pineapple", "pomegranate"))
print(letter_either_not_both("strawberry", "litchi"))
```

```
['a', 'e', 'g', 'l', 'm', 'n', 'o', 'p']
['a', 'e', 'n', 'p']
['a', 'b', 'c', 'e', 'h', 'i', 'l', 'r', 's', 'w', 'y']
```

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).

[3]:
```
def manage_country_capitals():
    country_capitals = {}

    while True:
        country = input("Enter the name of 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 of {country} is␣
 ↪{country_capitals[country_lower]}.")
        else:
            capital = input(f"Enter the capital city of {country}: ").strip()
            country_capitals[country_lower] = capital
            print(f"Thank you! The capital of {country} has been recorded as␣
 ↪{capital}.")

manage_country_capitals()
```

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

Thank you! The capital of Nepal has been recorded as Kathmandu.

Enter the name of a country (or type 'exit' to quit):  Switzerland
Enter the capital city of Switzerland:  Bern

Thank you! The capital of Switzerland has been recorded as Bern.

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

The capital of Nepal is Kathmandu.

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

Exiting the program.

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

Goodbye!
```

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.

[13]:
```
def report_most_common_letters(message):
    letters = [char.lower() for char in message if char.isalpha()]
    letter_counts = {}
    for letter in letters:
        letter_counts[letter] = letter_counts.get(letter, 0) + 1

    sorted_counts = sorted(letter_counts.items(), key=lambda x: (-x[1], x[0]))
    return sorted_counts[:6]


print(report_most_common_letters("This message will report most common␣
 ↪letters"))
```

```
[('e', 5), ('s', 5), ('t', 5), ('m', 4), ('o', 4), ('l', 3)]
```

[ ]:
```

```