Week7_programs

January 16, 2025

[12]:
```
'''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_letters(word):
    return list(sorted(set(word)))

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

main()
```

```
Enter a word:  tangerine
```

```
Unique sorted letters: ['a', 'e', 'g', 'i', 'n', 'r', 't']
```

[17]:
```
'''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 union(w1, w2):
    return list(sorted(set(w1) | set(w2)))

def intersection(w1, w2):
    return list(sorted(set(w1) & set(w2)))

def symmetric_difference(w1, w2):
    return list(sorted(set(w1) ^ set(w2)))

def main():
    w1 = "switch"
    w2 = "stick"

    print(f"Letters that appear in at least one of the two words: {union(w1,␣
  ↪w2)}")
    print(f"Letters that appear in both words: {intersection(w1, w2)}")
    print(f"Letters that appear in either word, but not in both:␣
  ↪{symmetric_difference(w1, w2)}")

main()
```

```
Letters that appear in at least one of the two words: ['c', 'h', 'i', 'k', 's',
't', 'w']
Letters that appear in both words: ['c', 'i', 's', 't']
Letters that appear in either word, but not in both: ['h', 'k', 'w']
```

[24]:
```
'''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.'''


countries = {
    'nepal': 'kathmandu',
    'india': 'new delhi',
    'france': 'paris',
    'japan': 'tokyo',
    'usa': 'washington dc',
    'canada': 'ottawa',
    'germany': 'berlin',
    'spain': 'madrid',
    'italy': 'rome',
    'brazil': 'brasilia'
}

def main():
    while True:
        country = input("Enter the country (or type 'exit' to quit): ")
        if country == 'exit':
            print("Goodbye!")
            break
        country_lower = ''
        for char in country:
            country_lower += char.lower()

        if country_lower in countries:
            print(f"The capital of {country} is {countries[country_lower]}")
        else:
            capital = input("Please enter the capital of" + country + ":")
            countries[country_lower] = capital
            print(f"Thank you! The capital of {country} is now {capital}.")

main()
```

```
Enter the country (or type 'exit' to quit):  Spain

The capital of Spain is madrid

Enter the country (or type 'exit' to quit):  brazil

The capital of brazil is brasilia

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

Goodbye!
```

[5]:
```
'''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.'''

def letter_frequency():
    message = input("Enter the encrypted message: ")
    message = message.lower()
    letter_counts = {}

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

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

    top_6 = sorted_letters[:6]

    print("The 6 most common letters are:")
    for letter, count in top_6:
        print(f"{letter}: {count}times")
letter_frequency()
```

```
Enter the encrypted message:  aenxysaeixsarsxpey
```

```
The 6 most common letters are:
s: 3times
x: 3times
e: 3times
a: 3times
y: 2times
p: 1times
```