DA108 | Lab 06 Assignment

Objective A:

Create an interactive command-line interface (CLI) application using python that: **translates words into emojis (including auto-filling mood-based emojis).**

Learn more before you start:

- What is unicode? https://youtu.be/ut74oHojxqo?si=4VGWrPyBHEovF2A1
- Not convinced about the importance of Unicode? Check this out - https://youtu.be/MijmeoH9LT4?si=wwA_ZulClMipkJxd

Step 1 - Building a Basic Emoji Translator:

Users input a sentence, and words are replaced with corresponding emojis. Below is a toy implementation.

Example dictionary:

```
emoji_dict = {
    "happy": "😀",
    "love": "❤️",
    "fire": "🔥",
    "sad": "😢",
    "cool": "😎",
    "food": "🍔",
    "dog": "🐶",
    "cat": "😺",
}
```

Basic Implementation:

```
def emoji_translate(sentence):
    words = sentence.split()
    return " ".join(emoji_dict.get(word, word) for word in words)

print(emoji_translate("I am happy and in love"))
# Output: I am 😀 and in ❤️
```

Now scale the above dictionary by populating with lot more elements inside the dictionary

Step 2 - Mood-Based Auto-Filling Emoji Generator:

Users can **request** a "mood fill" option by specifying a mood (e.g., happy, sad, excited).

- The system will **append relevant emojis** automatically.
    - Example:
        - **User Input:** `"Hey, how are you doing? (fill with happy emojis)"`
        - **Output:** `"Hey, how are you doing? 😃🎉🥳😊🌞"`

Can you think how this can be implemented? Below is a toy example.

First, create a dictionary mapping moods to a list of relevant emojis:

```
mood_emojis = {
    "happy": ["😀", "🎉", "😻", "😊", "🤩"],
    "sad": ["😳", "💔", "😭", "😞"],
    "excited": ["😻", "🔥", "🎊", "🚀"],
    "angry": ["😡", "💢", "👿", "🤬"],
}
```

*Next, detect if the user requests a **mood fill**, then append random emojis from the list:*

```
import random

def mood_fill(sentence):
    words = sentence.split()
    mood = None

    # Detect mood request
    for key in mood_emojis:
        if f"(fill with {key} emojis)" in sentence:
            mood = key
            words.remove(f"(fill with {key} emojis)")
            break

    # Append random emojis if mood detected
    if mood:
        words.append("".join(random.choices(mood_emojis[mood], k=3)))

    return " ".join(words)

print(mood_fill("Hey, how are you doing? (fill with happy emojis)"))
# Output: Hey, how are you doing? 😀🎉😊
```

Step 3 - Now, integrating the above features into one code. When you run your program on your terminal it should allow the user by helping with the following tasks.

- Translate sentences into emojis.
- Auto-fill emojis based on mood.

To accomplish the above, your code block should build upon the following structure.

```
while True:
    print("\n🎉 Welcome to Emoji Translator 🎉")
    print("1. Emoji Translator")
    print("2. Mood-Fill Emoji Generator")
    print("5. Exit")

    choice = input("Choose an option (1-5): ")

    if choice == "1":
        text = input("Enter a sentence to translate: ")
        print("📝 Translated:", emoji_translate(text))
    elif choice == "2":
        text = input("Enter a sentence with mood (e.g., 'I love coding! (fill with happy emojis)'): ")
        print("😊 Mood-Filled:", mood_fill(text))
    elif choice == "5":
        print("Goodbye! 👋")
        break
    else:
        print("Invalid choice! Try again.")
```

Objective B: Language Detection:

Create a small text sample (~50 sentences) by copy pasting translations from the following link. Note the 50 sentences should be drawn from multiple languages (e.g., English, Russian, Hindi, Chinese, Arabic, Spanish, French, German, Japanese, Korean, and more!). Let the resulting file.txt you thus created be named as file.txt and this resides in your system.

Link: https://tatoeba.org/en/

Next, complete the following task.

- Load the text file (`open("file.txt", "r")`).
- Detect language (or script) usage (Latin, Cyrillic, Arabic, etc.).
- Count words per script.
- Print a summary of how many different languages are present.

Below is a toy example code. Note the usage of the unicode package to help accomplish the task.

```
import unicodedata

def detect_script(char):
    """Identify script of a character"""
    try:
        return unicodedata.name(char).split()[0]
    except ValueError:
        return "UNKNOWN"

def analyze_text_file(filename):
    """Analyze script distribution in a text file"""
    with open(filename, "r", encoding="utf-8") as f:
        text = f.read()

    script_counts = {}
    for char in text:
        if char.isalpha():
            script = detect_script(char)
            script_counts[script] = script_counts.get(script, 0) + 1

    return script_counts

# Example usage
file_results = analyze_text_file("multilingual_text.txt")
print(file_results)
```