PYTHON.md

wordgenerator.py

Introduction

The Python backend of the Hangul Learning App is responsible for generating random Hangul words and Romanizing them for pronunciation practice. This section provides an in-depth look into the Python code, including installation instructions for Mac and Linux, library dependencies, and a detailed explanation of the key components.

Python Installation

For Mac:

1. **Install Homebrew (if not already installed):**

```
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

2. **Install Python:**

```
brew install python
```

3. **Install required Python packages:**

```
pip install flask pandas korean-romanizer
```

For Linux:

1. **Install Python:**

```
sudo apt update
sudo apt install python3
```

2. **Install pip (if not already installed):**

```
sudo apt install python3-pip
```

3. **Install required Python packages:**

```
pip3 install flask pandas korean-romanizer
```

Certainly! Let's break down the Python code into smaller chunks and explain each part:

Part 1: Imports

```
from flask import Flask, request, jsonify
import random
import pandas as pd
from korean_romanizer.romanizer import Romanizer
```

- **Explanation:**
  - Importing necessary modules and libraries for the web application.
  - `Flask`: A web framework for building the backend of the application.
  - `request`: Used to handle HTTP requests.
  - `jsonify`: Converts Python objects into JSON format for HTTP responses.
  - `random`: Provides functions for generating random numbers.
  - `pandas`: A library for data manipulation and analysis.
  - `Romanizer`: A class from the `korean-romanizer` library for romanizing Hangul.

Part 2: Flask App Setup

```
app = Flask(__name__)
```

- **Explanation:**
  - Creating a Flask application instance named `app`.

Part 3: Helper Functions

```
def csv_arr(path):
    df = pd.read_csv(path)
    return df['Syllable'].to_numpy()

def generate_word(syllables, length_word):
    random_syllables = random.choices(syllables, k=length_word)
    return ''.join(random_syllables)
```

```
def romanize_hangul(word):
    if not isinstance(word, str):
        raise ValueError('Input must be a string')

    romanizer = Romanizer(text=word)
    romanized_word = romanizer.romanize()
    return romanized_word
```

- **Explanation:**
  - `csv_arr`: Reads a CSV file at a given path, extracts the 'Syllable' column, and returns it as a NumPy array.
  - `generate_word`: Creates a random Hangul word of a specified length based on the provided syllables.
  - `romanize_hangul`: Uses the `korean-romanizer` library to romanize a Hangul word.

Part 4: Flask

```
@app.route('/generate_word', methods=['GET'])
def generate_word_api():
    try:
        length_word = int(request.args.get('length_word'))
        difficulty = int(request.args.get('difficulty'))

        if difficulty ≤ 0:
            return jsonify({'error': 'Difficulty must be a positive integer'}),
400

        if length_word ≤ 0:
            return jsonify({'error': 'Number of syllables must be a positive
integer.'}), 400

    except ValueError:
        return jsonify({'error': 'Invalid input. Please provide a positive
integer for the number of syllables.'}), 400

    if difficulty == 1:
        path = './difficulty1.csv'
    elif difficulty == 2:
        path = './difficulty2.csv'
    elif difficulty == 3:
        path = './difficulty3.csv'
    elif difficulty == 4:
        path = './difficulty4.csv'
    elif difficulty == 5:
        path = './difficulty5.csv'

    syllables = csv_arr(path)
```

```
    random_korean_word = generate_word(syllables, length_word)
    romanized_korean_word = romanize_hangul(random_korean_word)

    return jsonify({'korean_word': random_korean_word, 'romanized_word':
romanized_korean_word})
```

- **Explanation:**
  - `@app.route('/generate_word', methods=['GET'])`: Defines a route for handling GET requests at the `/generate_word` endpoint.
  - The function `generate_word_api` extracts two parameters (length and difficulty).
  - Based on the difficulty the program will handle different files and then generate a random Korean word.
  - Then the generated word will be Romanized.
  - The generated word and its Romanized version are returned as a JSON response.

Part 5: Server Execution

```
if __name__ == "__main__":
    app.run(port=4848)
```

- **Explanation:**
  - The application runs only if the script is executed directly, not imported as a module.
  - The Flask app is run on port `4848`.

Generator.py

Introduction

This app has been used to generate `5 csv files` with all the possible combination of Korean letters in a single syllable. To know more about all the levels consult the file KOREAN-HANGUL.md.

Full Code

```
import hangul_jamo as jamo
import csv

# Define consonants, vowels, and final consonants
initial_consonants_easy = ['ㄱ', 'ㄴ', 'ㄷ', 'ㄹ', 'ㅁ', 'ㅂ', 'ㅅ', 'ㅇ']
initial_consonants = ['ㄱ', 'ㄴ', 'ㄷ', 'ㄹ', 'ㅁ', 'ㅂ', 'ㅅ', 'ㅇ', 'ㅈ', 'ㅊ',
'ㅋ', 'ㅌ', 'ㅍ', 'ㅎ']
medial_vowels_easy = ['ㅏ', 'ㅑ', 'ㅓ', 'ㅕ', 'ㅗ', 'ㅛ', 'ㅜ', 'ㅠ', 'ㅡ', 'ㅣ']
```

```
medial_vowels = ['ㅏ', 'ㅑ', 'ㅓ', 'ㅕ', 'ㅗ', 'ㅛ', 'ㅜ', 'ㅠ', 'ㅡ', 'ㅣ', 'ㅐ',
'ㅒ', 'ㅔ', 'ㅖ', 'ㅘ', 'ㅙ', 'ㅚ', 'ㅝ', 'ㅞ', 'ㅟ', 'ㅢ']
final_consonants_easy = ['ㄱ', 'ㄴ', 'ㄷ', 'ㄹ', 'ㅁ', 'ㅂ', 'ㅅ', 'ㅇ', 'ㅈ', 'ㅊ',
'ㅋ', 'ㅌ', 'ㅍ', 'ㅎ']
final_consonants = [' ', 'ㄱ', 'ㄲ', 'ㄳ', 'ㄴ', 'ㄵ', 'ㄶ', 'ㄷ', 'ㄹ', 'ㄺ', 'ㄻ',
'ㄼ', 'ㄽ', 'ㄾ', 'ㄿ', 'ㅀ', 'ㅁ', 'ㅂ', 'ㅄ', 'ㅅ', 'ㅆ', 'ㅇ', 'ㅈ', 'ㅊ', 'ㅋ',
'ㅌ', 'ㅍ', 'ㅎ']


diff_1 = []
for letter in initial_consonants_easy:
    diff_1.append(letter)

for letter in medial_vowels_easy:
    diff_1.append(letter)


diff_2 = []
for letter in final_consonants:
    diff_2.append(letter)

for letter in medial_vowels:
    diff_2.append(letter)


diff_3 = []
for initial in initial_consonants:
    for medial in medial_vowels:
        syllable = initial + medial
        jm_syllable = jamo.compose(f'{syllable}')
        diff_3.append(jm_syllable)


diff_4 = []
for syllable in diff_3:
    diff_4.append(syllable)

for initial in initial_consonants:
    for medial in medial_vowels:
        for final in final_consonants_easy:
            syllable = initial + medial + final
            jm_syllable = jamo.compose(f'{syllable}')
            diff_4.append(jm_syllable)


diff_5 = []  # Generate triple syllables
for initial in initial_consonants:
    for medial in medial_vowels:
        for final in final_consonants:
```

```
            syllable = initial + medial + final
            jm_syllable = jamo.compose(f'{syllable}')
            diff_5.append(jm_syllable)

for initial in initial_consonants:
    for medial in medial_vowels:
        syllable = initial + medial
        jm_syllable = jamo.compose(f'{syllable}')
        diff_5.append(jm_syllable)


# Write syllables to a CSV file
with open('difficulty1.csv', 'w', newline='', encoding='utf-8') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(['Syllable'])
    for syllable in diff_1:
        writer.writerow([syllable])

# Write syllables to a CSV file
with open('difficulty2.csv', 'w', newline='', encoding='utf-8') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(['Syllable'])
    for syllable in diff_2:
        writer.writerow([syllable])

# Write syllables to a CSV file
with open('difficulty3.csv', 'w', newline='', encoding='utf-8') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(['Syllable'])
    for syllable in diff_3:
        writer.writerow([syllable])

# Write syllables to a CSV file
with open('difficulty4.csv', 'w', newline='', encoding='utf-8') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(['Syllable'])
    for syllable in diff_4:
        writer.writerow([syllable])

# Write syllables to a CSV file
with open('difficulty5.csv', 'w', newline='', encoding='utf-8') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(['Syllable'])
    for syllable in diff_5:
        writer.writerow([syllable])


print(f"Generated {len(diff_1) + len(diff_2) + len(diff_3) + len(diff_4) +
len(diff_5)} Hangul syllables and saved them to 'hangul_syllables.csv'.")
```