Lab 7

Due date: Mar 7, 2020, 4:00 am

Overview

In this lab, you will implement a simple RESTful API that provides synonyms of English words. You will practice common Python programming features like file operations, dictionaries, lists, string operations, conditional statements, loops, and function definitions.
You will also go through the setup process for the starter Flask application.

Routing and RESTful APIs

The resource section on the course website gives you short introduction to how Flask applications work and into RESTful APIs. We will revisit the basics of Flask, routing, and RESTful APIs in more detail next week.

Getting started

Before working on the Python code, we will go through the entire setup process to get the Python Flask project ready for further development. In future, we will simplify some steps.

- Import the starter project into Eclipse.
- Rename the project to `lab7`*`FirstnameLastname`* where *`FirstnameLastname`* is your first and last name.
- Open a local terminal and navigate to the project folder `lab7`*`FirstnameLastname`*. You can open the terminal through Eclipse, but you can also open the Windows PowerShell or Command Prompt, for example.
- In the terminal, setup a virtual environment for your project, called `venv`, in order to work with Python Flask:

  ```
  python -m venv venv
  ```

- Activate the virtual environment. On Linux or Mac OS, use the command:

  ```
  . venv/bin/activate
  ```

  on Windows use the command:

  ```
  venv\Scripts\activate
  ```

- Install Flask:

  ```
  python -m pip install flask
  ```

- Install python-dotenv:

  ```
  python -m pip install python-dotenv
  ```

- Create the file `.flaskenv` with the content:

  ```
  FLASK_ENV=development
  FLASK_APP=app\app.py
  ```

- In Eclipse, add the python interpreter from the above created virtual environment and set the new interpreter for your project. (See Lab 6, Section “Working with Flask and Virtual Environments”, Steps 7-10)
- In the terminal, start Flask:

  ```
  flask run
  ```

- Now you can test the starter project:
  - Open the URL `http://127.0.0.1:5000/synonyms` in a web browser. The browser will display the content of the variable `all_synonyms` as set by the function `get_all_synonyms()` in `app.py`. The variable `all_synonyms` is a dictionary with three entries. The key of each entry in the dictionary is a string. The value of the key is a list of synonyms of that word representing the key. Thus, for example, the key “eat” has the synonyms "consume", "use up", "deplete", "exhaust", and "feed".
  - Now open the URL `http://127.0.0.1:5000/synonym/eat`. The browser will display the synonyms of eat. Analogously, the browser will display the synonyms of “family” when opening `http://127.0.0.1:5000/synonym/family`, and the synonyms of “house” when opening http://127.0.0.1:5000/synonym/house. If the last part in the URL is a word that is not a key in the dictionary, say the word `cat`, then the application returns the message “No synonyms available for cat”.

Tasks

The starter application is an API that provides a list of words with their synonyms and synonyms for a given word. The synonyms known by the application are hard-coded in the Python program. You will modify the functionality to read the synonyms from a file instead of hard-coding a dictionary.

1. In the `app` source folder, add a new Python module, called `synonyms.py`.
2. Define a function called `get_synonyms_from_files`.
   a. For now, copy the code from function `get_all_synonyms()` in `app.py` into the new function:

   ```
   def get_synonyms_from_file():
       all_synonyms = {"family": ["class" , "category", "folk", "kinsfolk", "fellowship"],
           "eat": ["consume", "use up", "deplete", "exhaust", "feed"],
           "house": ["household", "home", "firm", "mansion", "put up", "domiciliate"]}
       return all_synonyms
   ```

   b. In `app.py`, initialize the variable `all_synonyms` in the view functions `get_all_synonyms` and `get_synonym(word)` to the return value of the new function `get_synonyms_from_file`. You will need to import the function from your new module `synonyms`:

```python
from flask import Flask
from synonyms import get_synonyms_from_file

app = Flask(__name__)


@app.route('/synonyms')
def get_all_synonyms():
    all_synonyms = get_synonyms_from_file()
    return all_synonyms

@app.route('/synonym/<string:word>')
def get_synonym(word):
    all_synonyms = get_synonyms_from_file()
    if word in all_synonyms.keys():
        return {word: all_synonyms[word]}
    else:
        return {"message": "No synonyms available for " + word}
```

c. Test the application and make sure everything works as before.

3. Now modify function `get_synonyms_from_file` to read the synonyms from the file `synonyms.txt`. As before, function `get_synonyms_from_file` should return a dictionary with synonyms. Each dictionary entry maps a word/phrase, the key, to a list of words/phrases, which are the synonyms of the key. File `synonyms.txt` is a simple text file where each line contains a pair of words or phrases that are synonyms. The words or phrases are separated by a comma. Each word may be preceded or followed by one or more spaces. Here are some valid example lines:

```
house,mansion
eat,consume
  family ,house
family, class
a bit,a little
a couple of, a few
```

Important: Flask runs the application at the project folder level `lab7FirstnameLastname`. Therefore, when opening file `synonyms.txt` (which is located in folder `lab7FirstnameLastname`) from a module in the `app` package, the relative path is just ”`synonyms.txt`”, not ”`../synonyms.txt`”.

Your solution has to meet the following requirements:

- To open a file, use the “with”-statement so that the file is guaranteed to be closed.
- If an exception occurs when reading from the file, catch the exception and return an empty dictionary. Print an error message to the console.
- If a line in file `synonyms.txt` has no valid data, e.g. it is an empty line or a single string, then the line is skipped and a message is printed to the terminal with the line number of the invalid line and the line content. For example, if line 14 of `synonyms.txt` has content is “house family” (note the missing comma), then the following message may be printed:

```
Error in line 10: house family
```

- Your solution will have to extract the two words or phrases from each line. All leading and trailing spaces should be removed from the word/phrase. Search online for suitable Python string operations.
- For each pair of words or phrases (`word1`, `word2`) extracted from a line, `word2` has to be added to the list of synonyms for `word1`, and `word1` has to be added to the list of synonyms for `word2`. The list of synonyms for a word should never contain duplicates. If there is not yet a dictionary entry with the key `word1`, add a new dictionary entry that maps `word1` to the list containing the singly element `word2`. Analogously, create a new dictionary entry for `word2` if the dictionary does not yet contain the key `word2`.

  For example, a file with the following content:

  ```
  house, mansion
  eat, consume
  consume, eat
  family, house
  family, class
  ```

  would result in the following dictionary:

  ```
  {
    “house”: [“mansion”, “family”],
    “mansion”: [“house”],
    “eat” : [consume],
    “consume”: [“eat”],
    “family”: [“house”, “class”],
    “class”: [“family”]
  }
  ```
- If your function gets too large, create additional functions that complete subtasks. As a rule-of-thumb, a function should not exceed 35 lines.

Submission

Archive the project folder `lab7`*`FirstnameLastname`* and submit the archived folder as `lab7`*`FirstnameLastname`*`.zip` to the course website. The archived project should contain the all files and folders in folder `lab7`*`FirstnameLastname`*, except for the folder `venv` with your virtual environment.