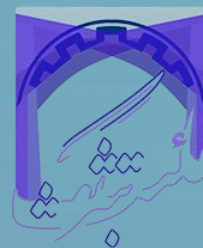


مکتب شریف

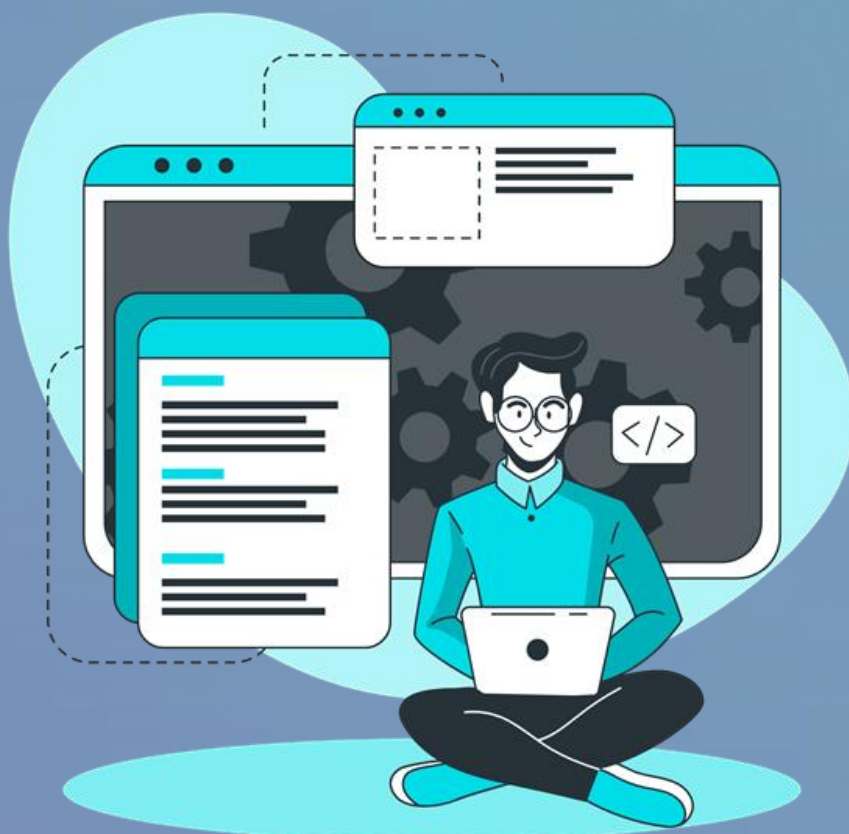
اولین بوتکمپ آموزشی - استخدامی ایران

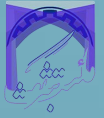


PYTHON

BOOTCAMP


PRACTICE 10






Weather Server Project

Introduction

Welcome to the Weather Server Project!  This project is designed to help you practice your Python skills by creating a simple weather server that retrieves weather data from an external API and displays it to clients.

 Your task is to implement the server and client commands and store data in a database.

Instructions

1. Run the server

To run the server, open a command line window and navigate to the project directory. Then, execute the following command:

```
python weather_server.py
```

2. Run the client command-line interface

In a separate command line window, navigate to the project directory and run the following command to start the client:

```
python weather_client.py
```

3. Enter a city name

Once the client is running, you will be prompted to enter a city name. Type the name of a city and press Enter to request weather data from the server:

```
Enter a city name: San Francisco
```

4. Display weather data

If the city name is valid, the weather information will be displayed on the command line, like this:

```
Temperature: 18.5°C  
Feels like: 16.3°C  
Last updated: 2023-06-22 15:40:00
```

5. Retry for an incorrect or invalid city name

If the city name entered is incorrect or invalid, an error message will be displayed. You can retry by entering a new city name:

```
Enter a city name: Invalid City  
Error retrieving weather data: No matching location found.  
Enter a new city name: San Francisco
```

6. Unit Testing

Add some unit tests for the `get_city_weather` method.

7. Entity Relationship Diagram (ERD)

Create an ERD to store request and response data in two tables.

8. Additional Methods 📊

Add the following methods to get:

- Count of request
- Count of successful requests
- Last hour requests
- Each city request count

9. Store Data in a Database 📄

Store the request and response data in a database based on your ERD.

10. Database Unit Testing 🔑

Add unit tests for your database methods.

Sample Output 📄

Here is a sample output of the completed project:

```
Enter a city name: San Francisco
Temperature: 18.5°C
Feels like: 16.3°C
Last updated: 2023-06-22 15:40:00

Enter a city name: Invalid City
Error retrieving weather data: No matching location found.

Request count: 2
Successful request count: 1
Last hour requests:
[('San Francisco', '2022-01-01 12:00:00'), ('Invalid City', '2022-01-01 12:15:00')]
City request counts:
[('San Francisco', 1), ('Invalid City', 1)]
```

Folder Structure

```
weather_project/
├── weather_server.py
├── weather_client.py
├── database.py
├── erd.png
├── requirements.txt
├── tests/
│   ├── test_get_city_weather.py
│   └── test_database.py
```

Function Definition

1. weather_server.py

```
def get_city_weather(city_name: str) -> dict:
    """
    Retrieve weather data from an external API for a given city.

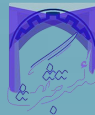
    Args:
        - city_name (str): The name of the city to retrieve weather data for.

    Returns:
        - dict: A dictionary containing weather information for the city, including temperature, feels like temperature, and last updated time.
    """
    pass

def start_server() -> None:
    """
    Start the weather server.
    """
    pass
```

2. weather_client.py

```
def start_client() -> None:
    """
    Start the weather client command-line interface.
    """
    pass
```



3. database.py

```

from typing import List, Tuple

class WeatherDatabase:
    def __init__(self):
        """
        Initialize a new WeatherDatabase instance.
        """
        pass

    def save_request_data(self, city_name: str, request_time: str) -> None:
        """
        Save request data for a city to the database.

        Args:
        - city_name (str): The name of the city to save request data for.
        - request_time (str): The time the request was made, in ISO format.

        Returns:
        - None
        """
        pass

    def save_response_data(self, city_name: str, response_data: dict) -> None:
        """
        Save response data for a city to the database.

        Args:
        - city_name (str): The name of the city to save response data for.
        - response_data (dict): A dictionary containing weather information for
        or the city, including temperature, feels like temperature, and last updated
        time.

        Returns:
        - None
        """
        pass

    def get_request_count(self) -> int:
        """
        Get the total number of requests made to the server.

        Returns:
        - int: The total number of requests made to the server.
        """
        pass

    def get_successful_request_count(self) -> int:
        """
        Get the total number of successful requests made to the server.

```

نکات

- مهلت ارسال تمرین تا **پایان ساعت ۲۴ روز چهارشنبه** می باشد
- زین پس تمامی تحویل تمرین تنها و تنها از طریق گیتهاب Github صورت می پذیرد.
- یک دارکتوری با نام **week13** در ریپازیتوری **HW-100** خود ایجاد کنید و پاسخ تمرین را درون آن قرار دهید. توجه داشته باشید که این ریپازیتوری باید **private** باشد.
- ملاک و معیار ارزیابی تاریخ آخرین **commit** شما می باشد. (بصورت استاندارد و اصولی کامیت انجام شود).
- **بخشی از نمره ی شما مربوط به کامیت گذاری صحیح است.**
- **در صورتی که سوالی دارید در کارتابل گروهی خود از مربیان بپرسید.**
- **توصیه دوستانه:** از مواجهه با هیچ سوالی نترسید. به هر میزانی که در حل سوالات پیشروی کرده باشید نمره بخش مورد نظر را دریافت می کنید. بنابراین بیش از آنکه رسیدن به خروجی نهایی مهم باشد، تلاش شما ارزشمندتر است.
- قطعا هدف از تمرین صرفا رسیدن به جواب نهایی نیست و تمیز بودن کد و خلاقیتی که در انجام آن به خرج می دهید از اهمیت و امتیاز بالایی برخوردار است. ارائه راه حل کلی و عمومی برای یک مسئله که حالت های مختلف آن را در نظر بگیرد و فراتر از خواسته ی مسئله است. (خواسته ی مسئله گسترش داده شود یا حالت های خاص مسئله را پوشش دهد. قطعا مشمول امتیاز بیشتری خواهد شد).

موفق باشید