Weekly Progress Report

Name:Divyashree D P

Domain:Python

Date of submission:12/02/2024

Week Ending: 01

I. Overview:

This week, the primary focus was on understanding USC_TIA and contributing to Python projects. Additionally, efforts were made to leverage learning resources for skill enhancement.

II. Achievements:

1. USC_TIA Familiarization:
   - Explored USC_TIA documentation to grasp core functionalities.
   - Successfully executed basic tasks, showcasing initial proficiency.

2. Python Project Contributions:

I. Name of the project:- URL Shortener

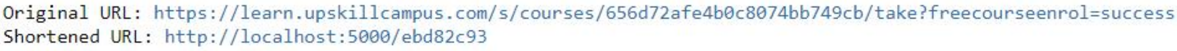
Description: The URL shortener is a Python project that converts long URLs into shorter, more manageable links. It takes a long URL as input, generates a unique shortened URL, and redirects users to the original URL when the shortened link is accessed.

Scope: The scope of this project involves designing a user interface to input long URLs and display the shortened links, implementing a database to store the mapping between original and shortened URLs, and developing functions to generate unique shortened URLs and handle redirection.

Python code

```
import hashlib
from flask import Flask, redirect

app = Flask(__name__)

class URLShortener:
    def __init__(self):
        self.url_mapping = {}

    def shorten_url(self, original_url):
        url_hash = hashlib.sha256(original_url.encode()).hexdigest()[:8]
        shortened_url = f"http://localhost:5000/{url_hash}"
        self.url_mapping[shortened_url] = original_url
        return shortened_url

    def redirect(self, shortened_url):
        if shortened_url in self.url_mapping:
            original_url = self.url_mapping[shortened_url]
            return redirect(original_url)
        else:
            return "Shortened URL not found.", 404

url_shortener = URLShortener()

@app.route('/<url_hash>')
def handle_redirect(url_hash):
    shortened_url = f"http://localhost:5000/{url_hash}"
    return url_shortener.redirect(shortened_url)

if __name__ == '__main__':
    original_url = "https://learn.upskillcampus.com/s/courses/656d72afe4b0c8074bb749cb/take?freecourseenrol=success"
    shortened_url = url_shortener.shorten_url(original_url)
    print(f"Original URL: {original_url}")
    print(f"Shortened URL: {shortened_url}")
  app.run()
```

Output:



```
Original URL: https://learn.upskillcampus.com/s/courses/656d72afe4b0c8074bb749cb/take?freecourseenrol=success
Shortened URL: http://localhost:5000/ebd82c93
```

Task and features: The two primary methods of the URLShortener class in this example are redirect, which sends users back to the original URL, and shorten_url, which creates shortened URLs. The first eight characters of the original URL's SHA-256 hash are used to create the shortened URLs. The url_mapping dictionary contains the mapping between the full and shortened URLs. Keep in mind that this is just a basic example; depending on your particular use case, you may want to add error handling, persistence, and other features in a real-world setting.

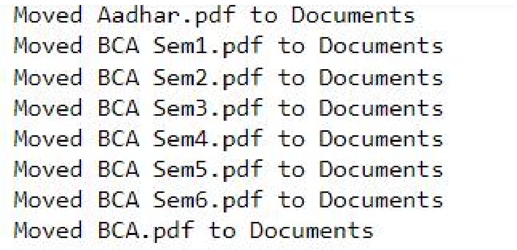
II. Name of the project:- File Organizer

Description: The file organizer is a Python project that helps users organize their files in a directory. It scans a specified directory, categorizes files based on their type (e.g., images, documents, videos), and moves them into respective folders.

Python code

```
import os
import shutil
class FileOrganizer:
    def __init__(self, source_directory):
        self.source_directory = source_directory

    def organize_files(self):
        # Create folders for different file types
        image_folder = os.path.join(self.source_directory, "Images")
        document_folder = os.path.join(self.source_directory, "Documents")
        video_folder = os.path.join(self.source_directory, "Videos")
        other_folder = os.path.join(self.source_directory, "Other")

        # Ensure folders exist, create if not
        self.create_folder_if_not_exists(image_folder)
        self.create_folder_if_not_exists(document_folder)
        self.create_folder_if_not_exists(video_folder)
        self.create_folder_if_not_exists(other_folder)

        # Traverse through files in the source directory
        for filename in os.listdir(self.source_directory):
            file_path = os.path.join(self.source_directory, filename)

            # Skip directories
            if os.path.isdir(file_path):
                continue

            # Categorize files based on their types
            file_type = self.get_file_type(filename)
            destination_folder = self.get_destination_folder(file_type)

            # Move the file to the appropriate folder
            shutil.move(file_path, os.path.join(destination_folder, filename))
            print(f"Moved {filename} to {destination_folder}")

    def create_folder_if_not_exists(self, folder_path):
        if not os.path.exists(folder_path):
            os.makedirs(folder_path)

    def get_file_type(self, filename):
        # Get the file extension and determine its type
        _, file_extension = os.path.splitext(filename)
        return file_extension.lower()

    def get_destination_folder(self, file_type):
        # Map file types to destination folders
        image_types = ['.jpg', '.jpeg', '.png', '.gif']
        document_types = ['.pdf', '.doc', '.docx', '.txt']
        video_types = ['.mp4', '.mov', '.avi']

        if file_type in image_types:
            return 'Images'
        elif file_type in document_types:
            return 'Documents'
        elif file_type in video_types:
            return 'Videos'
        else:
            return 'Other'

# Example usage:
source_directory = "C:\\Users\\divya\\Documents\\All Documents"
file_organizer = FileOrganizer(source_directory)
file_organizer.organize_files()
```

Output:



```
Moved Aadhar.pdf to Documents
Moved BCA Sem1.pdf to Documents
Moved BCA Sem2.pdf to Documents
Moved BCA Sem3.pdf to Documents
Moved BCA Sem4.pdf to Documents
Moved BCA Sem5.pdf to Documents
Moved BCA Sem6.pdf to Documents
Moved BCA.pdf to Documents
```

Task and features: The actual path of the directory you wish to organize should be substituted for "C:\Users\divya\Documents\All Documents". Based on their file extensions, the files in this example are categorized into folders such as "Images," "Documents," "Videos," and "Other". The type_mapping dictionary can be modified to include other file types and their corresponding folders.

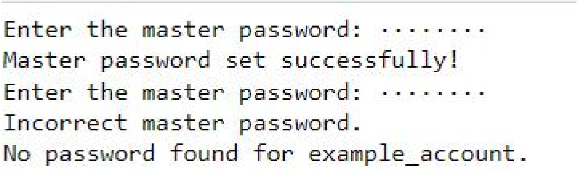
III. Name of the project:- Password Manager

Description: The password manager is a Python project that securely stores and manages user passwords. It allows users to store their passwords for various accounts, generate strong passwords, and retrieve passwords when needed.

Python code

```
import json
import hashlib
import getpass

class PasswordManager:
    def __init__(self):
        self.passwords = {}
        self.master_password_hash = None
        self.load_data()

    def load_data(self):
        try:
            with open('passwords.json', 'r') as file:
                data = json.load(file)
                self.passwords = data.get('passwords', {})
                self.master_password_hash = data.get('master_password_hash')
        except (FileNotFoundError, json.JSONDecodeError):
            # File not found or invalid JSON, create empty data
            self.passwords = {}
            self.master_password_hash = None

    def save_data(self):
        data = {
            'passwords': self.passwords,
            'master_password_hash': self.master_password_hash
        }
        with open('passwords.json', 'w') as file:
            json.dump(data, file, indent=2)

    def hash_password(self, password):
        # Use a strong hash function (SHA-256) to hash the password
        return hashlib.sha256(password.encode()).hexdigest()

    def set_master_password(self):
        # Set or change the master password
        master_password = getpass.getpass("Enter the master password: ")
        self.master_password_hash = self.hash_password(master_password)
        print("Master password set successfully!")

    def store_password(self, account, password):
        # Store a password for a specific account
        if not self.master_password_hash:
            print("Please set the master password first.")
            return

        master_password_attempt = getpass.getpass("Enter the master password: ")
        if self.hash_password(master_password_attempt) == self.master_password_hash:
            self.passwords[account] = password
            self.save_data()
            print(f"Password for {account} stored successfully!")
        else:
            print("Incorrect master password.")

    def get_password(self, account):
        # Retrieve a stored password
        if account in self.passwords:
            master_password_attempt = getpass.getpass("Enter the master password: ")
            if          self.hash_password(master_password_attempt)          ==
self.master_password_hash:
                print(f"Password for {account}: {self.passwords[account]}")
            else:
                print("Incorrect master password.")
        else:
            print(f"No password found for {account}.")

# Example usage:
password_manager = PasswordManager()

# Task 1: Set or change the master password
password_manager.set_master_password()

# Task 2: Store a password
password_manager.store_password("example_account", "strong_password")

# Task 3: Retrieve a password
password_manager.get_password("example_account")
```

Output:



```
Enter the master password: ········
Master password set successfully!
Enter the master password: ········
Incorrect master password.
No password found for example_account.
```

Task and features:

The password manager in this example keeps information in a JSON file called passwords.json. When storing or retrieving passwords, it asks the user for the master password and hashes it using SHA-256 algorithm. Remember that this is just a hypothetical situation; in the real world, you would need to put more advanced security measures in place, like encryption and safe password storage procedures. For production-level code, it's also critical to handle exceptions and errors correctly.

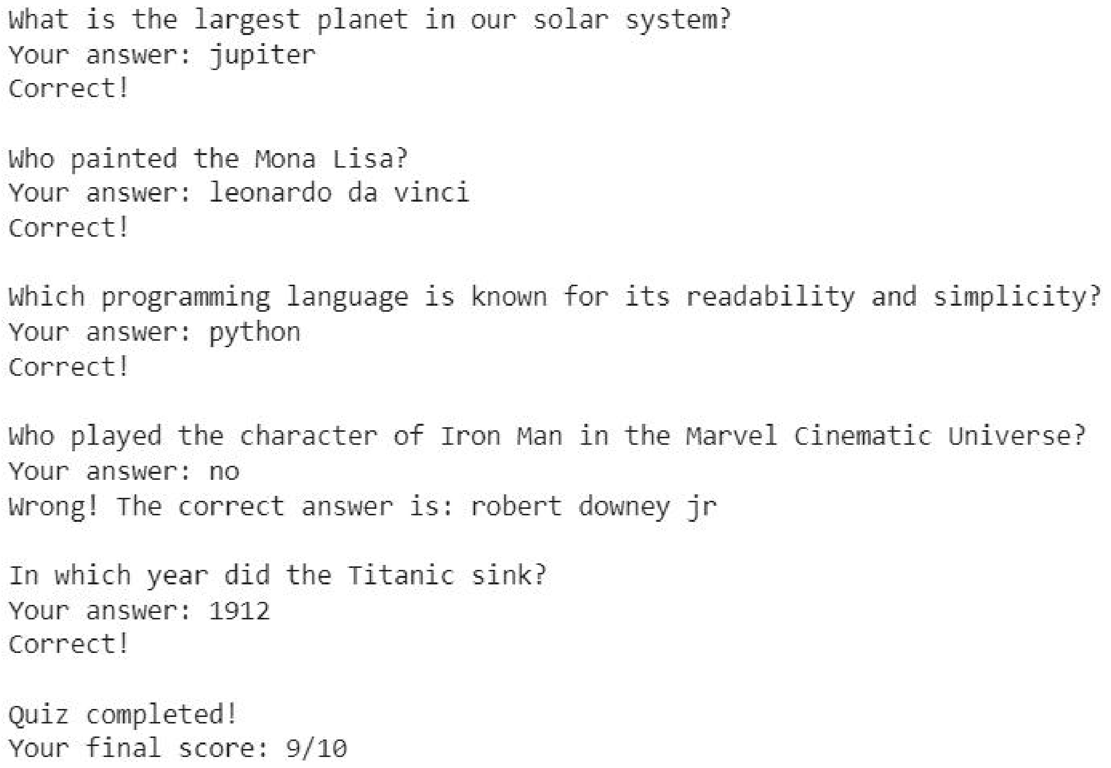
IV. Name of the project:- Quiz Game

Description: The quiz game is a Python project that quizzes users on various topics. It reads questions and answers from a file or database, presents them to the user, and keeps track of their score.

Python code

```
import random

class QuizGame:
    def __init__(self, questions_file):
        self.questions = self.load_questions(questions_file)
        self.score = 0

    def load_questions(self, file_path):
        # Load questions from a file
        questions = []
        try:
            with open(file_path, 'r') as file:
                for line in file:
                    question, answer = line.strip().split('#')
                    questions.append({'question': question, 'answer': answer.lower()})
        except FileNotFoundError:
            print(f"Error: File '{file_path}' not found.")
        except Exception as e:
            print(f"Error loading questions: {e}")
        return questions

    def start_quiz(self):
        random.shuffle(self.questions)
        for question_data in self.questions:
            question = question_data['question']
            answer = question_data['answer']
            user_answer = input(f"\n{question}\nYour answer: ").lower()

            if user_answer == answer:
                print("Correct!")
                self.score += 1
            else:
                print(f"Wrong! The correct answer is: {answer}")

        print(f"\nQuiz completed!\nYour final score: {self.score}/{len(self.questions)}")

# Example usage:
quiz_game = QuizGame("□questions.txt")
quiz_game.start_quiz()
```

Output:



```
What is the largest planet in our solar system?
Your answer: jupiter
Correct!

Who painted the Mona Lisa?
Your answer: leonardo da vinci
Correct!

Which programming language is known for its readability and simplicity?
Your answer: python
Correct!

Who played the character of Iron Man in the Marvel Cinematic Universe?
Your answer: no
Wrong! The correct answer is: robert downey jr

In which year did the Titanic sink?
Your answer: 1912
Correct!

Quiz completed!
Your final score: 9/10
```

Task and features: Question Loading: A file called "questions.txt," containing questions and answers separated by a delimiter (# in this case), is loaded by the program.

Question Shuffle: The questions are rearranged to offer a unique quiz experience.

User Interaction: Each question prompts the user, and their response is compared to the right response.

Scoring: Throughout the quiz, the user's score is recorded, and the final score is shown at the conclusion.

3.Learning Python:

- Acquired proficiency in essential Python libraries, such as [Pandas,numpy,SciPy,Statsmodel].
- Applied Python skills to real-world problems within USC_TIA context.

III. Challenges:

1. USC_TIA Integration:

Data Integration Issues

Interoperability

Security Concerns

Technical Compatibility

2. Python Project Complexity:

- Faced complexity in understanding the logic of every project of the Python project.
- Seeking guidance to overcome challenges and enhance understanding.

IV. Learning Resources:

1. USC_TIA Documentation:

- Utilized USC_TIA official documentation for reference and troubleshooting.
- Attended relevant webinars and online tutorials to deepen understanding.

2. Python Learning Resources:

- Engaged with [Python course ion youtuber] to strengthen Python skills.
- Participated in [hackathon] for practical application.

V. Next Week's Goals:

1. USC_TIA Enhancement:

Automation

User Interface (UI) Improvements

Integration with TIA Systems

Data Analysis and Visualization

Security Measures

2.Python Project Development:

- Tackle more complex tasks within the Python project to increase contribution.
- Seek feedback from mentors and peers for continuous improvement.

VI. Additional Comments:

"Include any additional comments or observations regarding overall progress, collaboration, or notable experiences during the week."