Weeked Assignment

Raghava Gatadi
21BCS088

Here is a Python code that can be used to create a new cultural destination in India and provide a platform for emerging talents using digital technology solutions:

```
# creating a function to create art space which saves detail about art
import unittest
import json
import os

def create_art_space(name, location, description):
    # Create a dictionary to hold the art space details
    art_space = {
        'name': name,
        'location': location,
        'description': description
    }

    # Open the file to store the art space details
    with open('art_spaces.json', 'a') as file:
        # Write the art space details to the file
        file.write(json.dumps(art_space))
        file.write('\n') # add a newline character to separate entries

    # Return a message indicating success
    return f'Art space {name} created successfully!'
```

```
class TestArtSpace(unittest.TestCase):

    def setUp(self):

        self.filepath = 'test_art_spaces.json'
        with open(self.filepath, 'w') as file:
            file.write('')

    def tearDown(self):

        os.remove(self.filepath)

    def test_create_art_space(self):

        name = 'Test Art Space'
        location = 'Test Location'
        description = 'Test Description'


        result = create_art_space(name, location, description)



        expected_output = f'Art space {name} created successfully!'
        self.assertEqual(result, expected_output)



        with open(self.filepath, 'r') as file:
            art_spaces = [json.loads(line) for line in file]
            self.assertEqual(len(art_spaces), 1)
            self.assertEqual(art_spaces[0]['name'], name)
            self.assertEqual(art_spaces[0]['location'], location)
            self.assertEqual(art_spaces[0]['description'], description)

#user management system
```

```
class User:
    def __init__(self, username, password, email):
        self.username = username
        self.password = password
        self.email = email
        self.profile = {}

    def create_profile(self, name, bio, location):
        self.profile = {"name": name, "bio": bio, "location": location}

class UserManagement:
    def __init__(self):
        self.users = {}

    def create_user(self, username, password, email):
        if username not in self.users:
            self.users[username] = User(username, password, email)
            return True
        else:
            return False

    def login(self, username, password):
        if username in self.users:
            user = self.users[username]
            if user.password == password:
                return user
        return None

    def update_profile(self, user, name=None, bio=None, location=None):
        if name:
            user.profile["name"] = name
        if bio:
            user.profile["bio"] = bio
        if location:
            user.profile["location"] = location

import unittest

class TestUserManagement(unittest.TestCase):

    def setUp(self):
        self.user_management = UserManagement()
        self.user1 = User("user1", "password1", "user1@example.com")
        self.user2 = User("user2", "password2", "user2@example.com")
        self.user_management.create_user("user1", "password1", "user1@example.com")

    def test_create_user(self):

        self.assertTrue(self.user_management.create_user("user2", "password2", "user2@example.com"))



        self.assertFalse(self.user_management.create_user("user1", "password1", "user1@example.com"))

    def test_login(self):

        self.assertEqual(self.user_management.login("user1", "password1"), self.user1)



        self.assertIsNone(self.user_management.login("user3", "password3"))



        self.assertIsNone(self.user_management.login("user1", "password2"))

    def test_update_profile(self):

        self.user_management.update_profile(self.user1, name="User One", bio="A bio", location="New York")
        self.assertEqual(self.user1.profile, {"name": "User One", "bio": "A bio", "location": "New York"})
        self.user_management.update_profile(self.user1, bio="Another bio")
        self.assertEqual(self.user1.profile, {"name": "User One", "bio": "Another bio", "location": "New York"})

def create_programming(name, description, category, date, location, performers):
    # logic to create a new programming item in the database
    programming_item = {
        'name': name,
        'description': description,
        'category': category,
        'date': date,
        'location': location,
        'performers': performers
    }
    # save the programming item in the database
    # return the ID of the new programming item
    return programming_item['id']

def get_programming(id):
    # logic to get a programming item from the database using its ID
    return programming_item

def update_programming(id, name=None, description=None, category=None, date=None, location=None,
performers=None):
    # logic to update a programming item in the database using its ID
    programming_item = get_programming(id)
    if name:
        programming_item['name'] = name
    if description:
        programming_item['description'] = description
    if category:
        programming_item['category'] = category
    if date:
        programming_item['date'] = date
    if location:
        programming_item['location'] = location
    if performers:
        programming_item['performers'] = performers
    # save the updated programming item in the database
    return programming_item

def delete_programming(id):
    # logic to delete a programming item from the database using its ID
    # return True if the item was deleted successfully, False otherwise
    return 'deleted_successfully'



import unittest

class TestProgrammingManagement(unittest.TestCase):

    def test_create_programming(self):
        result = create_programming('Test Programming', 'This is a test programming item', 'music', '2023-04-01', 'New
Delhi', ['performer1', 'performer2'])
        self.assertIsInstance(result, str)
        self.assertGreater(len(result), 0)

    def test_get_programming(self):
        id = create_programming('Test Programming', 'This is a test programming item', 'music', '2023-04-01', 'New
Delhi', ['performer1', 'performer2'])
        result = get_programming(id)
        self.assertIsNotNone(result)

    def test_update_programming(self):
        id = create_programming('Test Programming', 'This is a test programming item', 'music', '2023-04-01', 'New
Delhi', ['performer1', 'performer2'])
        result = update_programming(id, name='Updated Programming', location='Mumbai')
        self.assertIsNotNone(result)
        self.assertEqual(result['name'], 'Updated Programming')
        self.assertEqual(result['location'], 'Mumbai')

    def test_delete_programming(self):
        id = create_programming('Test Programming', 'This is a test programming item', 'music', '2023-04-01', 'New Delhi', ['performer1', 'performer2'])
        result = delete_programming(id)
        self.assertTrue(result)
```