LAB #1—Student Grades Tracker—Part 1

Overview

In this first assignment, you will build the foundation of a grade management system. You'll create a program that allows students to add courses and their corresponding grades, focusing on basic data structure manipulation and input/output operations.

Learning Objectives

By completing this assignment, you will:

1. Practice working with nested dictionaries
2. Implement input validation
3. Create organized functions for basic operations
4. Handle basic user interaction through a menu system

Requirements

1. Data Structure

Create a dictionary to store course information with the following structure:

```
{
 'COURSE_CODE': {
  'course_name': 'COURSE_NAME',
  'quizzes': [],
  'cats': [],
  'final_exam': None
 }
}
```

2. Required Functions

Implement the following functions:

1. `add_course(courses_dict, course_code, course_name)`
    - Adds a new course to the system
    - Returns the updated courses dictionary
2. `add_grade(courses_dict, course_code, grade_type, grade)`

  - Adds a grade to the specified course
  - grade_type can be 'quiz', 'cat', or 'final'
  - Returns the updated courses dictionary

3. `display_courses(courses_dict)`
   - Displays all courses and their grades
   - Shows course code, name, and all recorded grades

3. Input Validation

Implement the following validations:

- Course codes must be non-empty strings
- Grades must be between 0 and 100
- Course names must be non-empty strings

4. User Interface

Create a simple menu-driven interface with these options:

1. Add a new course
2. Add a grade to a course
3. Display all courses and grades
4. Exit

Helpful Hints

1. Data Structure Design

- Start by creating an empty dictionary to store all courses
- For each course, create a nested dictionary with empty lists for grades
- Example of adding a course:

```
courses['CS101'] = {
 'course_name': 'Intro to Programming',
 'quizzes': [],
 'cats': [],
 'final_exam': None
}
```

2. Function Implementation

- Break down each function into smaller steps
- Use `if` statements to validate inputs before processing
- Consider what should happen if a course code doesn't exist
- Think about how to handle invalid inputs gracefully

3. User Interface Tips

- Use a `while` loop for the main menu
- Clear validation messages help users understand what went wrong
- Consider using `try-except` blocks for number inputs

Sample Output

Your program should produce output similar to this:

```
Welcome to Grade Manager v1.0

1. Add a new course
2. Add a grade to a course
3. Display all courses and grades
4. Exit

Choice: 1
Enter course code: CS101
Enter course name: Introduction to Python
Course added successfully!

Choice: 2
Enter course code: CS101
Enter grade type (quiz/cat/final): quiz
Enter grade (0-100): 85
Grade added successfully!

Choice: 3
Course: CS101 (Introduction to Python)
Quizzes: [85]
CATs: []
Final Exam: None
```

Grading Criteria

- Correct implementation of data structure (25%)
- Proper function implementation (25%)
- Input validation (25%)
- User interface and output formatting (15%)
- Code organization and comments (10%)

Submission Guidelines

- Submit a single Python file named `grade_manager.py`
- Include comments explaining your code
- Ensure your program runs without any additional dependencies