# **SEED 2025: Python Training**

# 

# **Day 2 Assignments**

## **Assignment 1: Student Gradebook Manager**

This assignment requires you to build a console-based student gradebook management system using a dictionary to store student records. You will implement a menu-driven interface that allows users to perform various operations on the gradebook. This assignment will help you strengthen your understanding of:

* Dictionaries and nested dictionaries for structured data storage
* Functions for modularity and reusability
* Control flow (while loops for the menu, if/elif/else for menu choices)
* Error handling (try-except for input validation)
* Basic input/output (input(), print())

### **Assignment Details:**

Your "Student Gradebook Manager" should meet the following requirements:

1. **Gradebook Data Structure:** Use a dictionary to store student data. Each key will be a student’s name (string), and the value will be another dictionary with subject names as keys and grades as values (integer or float between 0 and 100).

gradebook = {

"Alice": {"Math": 95, "English": 88},

"Bob": {"Math": 78, "English": 85}

}

1. **Menu-Driven Interface:** The system should present a menu to the user with the following options:  
   * 1. View All Records: Display all students and their subjects with grades.
   * 2. Add New Student: Add a new student to the gradebook with no subjects initially.
   * 3. Add/Update Subject Grade: Add a new subject or update an existing subject grade for a student.
   * 4. Remove Student: Remove a student and their records from the gradebook.
   * 5. Calculate Average Grade: Calculate and display the average grade of a specific student.
   * 6. Exit: Terminate the program.
2. **Input Validation (try-except):**
   * When entering grades, ensure the input is a number between 0 and 100. Use try-except blocks to catch ValueError for non-numeric input.
   * If a student is to be updated or removed, check if they exist in the gradebook. If not, inform the user.
   * Ensure that names and subject inputs are non-empty strings.
3. **User-Friendly Messages:** Provide clear and helpful messages to the user after each operation, including confirmations of success, errors, or invalid input notifications.

## **Assignment 2: Simple Banking System**

This assignment requires you to build a console-based banking system using dictionaries to manage customer accounts. You will implement a menu-driven interface that allows users to perform common banking operations. This assignment will help reinforce your understanding of:

* Dictionaries and nested dictionaries for structured data storage
* Functions for modularity and reusability
* Control flow (while loops for the menu, if/elif/else for choices)
* Error handling (try-except for robust input validation)
* Basic input/output (input(), print())

### **Assignment Details:**

Your "Simple Banking System" should meet the following requirements:

1. **Bank Account Data Structure:** Use a dictionary to store bank account information. The keys will be account numbers (strings), and the values will be dictionaries containing the account holder’s name and balance.  
     
   accounts = {

"1001": {"name": "Alice", "balance": 5000.0},

"1002": {"name": "Bob", "balance": 3200.0}

}

1. **Menu-Driven Interface:** The system should present a menu to the user with the following options:  
   * 1. View All Accounts: Display account numbers, names, and balances.
   * 2. Create New Account: Add a new account by entering account number, name, and initial deposit.
   * 3. Deposit Money: Add a specified amount to an existing account’s balance.
   * 4. Withdraw Money: Deduct a specified amount from an account, ensuring sufficient balance.
   * 5. Check Balance: Display the balance of a specific account.
   * 6. Exit: Terminate the program.
2. **Input Validation (try-except):**
   * Ensure deposit and withdrawal amounts are valid floats and non-negative.
   * Use try-except blocks to catch invalid numeric input.
   * Check that an account exists before attempting deposits, withdrawals, or balance checks.
   * Prevent withdrawals that would result in a negative balance.
3. **User-Friendly Messages:** Display appropriate messages for successful transactions, errors (e.g., insufficient funds), and invalid menu options.

## **Assignment 3: Library Book Management System**

This assignment requires you to build a console-based system to manage a library’s book collection using a dictionary to track available books and their details. This assignment will help you practice:

* Dictionaries and nested dictionaries for storing book data
* Functions for organization and code reuse
* Control flow (while loops, if/elif/else statements)
* Error handling using try-except
* Basic input/output (input(), print())

### **Assignment Details:**

Your "Library Book Management System" should meet the following requirements:

1. **Library Data Structure:** Use a dictionary where each key is a book title (string), and each value is another dictionary containing the author's name and number of copies.  
     
   library = {

"1984": {"author": "George Orwell", "copies": 3},

"Dune": {"author": "Frank Herbert", "copies": 5}

}

1. **Menu-Driven Interface:** The system should present a menu to the user with the following options:  
   * 1. View All Books: Display all book titles, authors, and available copies.
   * 2. Add New Book: Add a new book to the library. If the book exists, update its copies.
   * 3. Borrow Book: Reduce the number of copies of a book by 1, if available.
   * 4. Return Book: Increase the number of copies of a book by 1.
   * 5. Remove Book: Remove a book from the library.
   * 6. Exit: Terminate the program.
2. **Input Validation (try-except):**
   * When entering the number of copies, ensure the input is a valid non-negative integer.
   * Use try-except blocks to catch invalid input and handle missing books gracefully.
   * Ensure that borrowing is only allowed if at least one copy is available.
3. **User-Friendly Messages:** Display informative messages for each operation, including success confirmations, error messages (e.g., trying to borrow a book that’s out of stock), and instructions for invalid input.

## **Assignment 4: Advanced Inventory Management System**

This assignment requires you to build a console-based inventory management system using a dictionary to store product information. You will implement a menu-driven interface, allowing users to perform various operations on the inventory. This assignment will help you solidify your understanding of:

* Dictionaries for data storage
* Functions for modularity and reusability
* Control flow (while loops for the menu, if/elif/else for menu choices)
* Error handling (try-except for robust input validation)
* Basic input/output (input(), print())

### **Assignment Details:**

Your **"Advanced Inventory Management System"** should meet the following requirements:

1. **Inventory Data Structure:** Use a dictionary to store inventory. The keys will be product names (strings), and the values will be lists containing [quantity, price].  
     
    inventory = {"Laptop": [10, 1200.00], "Mouse": [50, 25.50]}
2. **Menu-Driven Interface:** The system should present a menu to the user with the following options:  
   * 1. View Inventory: Display all products, their quantities, and prices.
   * 2. Add Product: Add a new product to the inventory. If the product already exists, update its quantity and price.
   * 3. Remove Product: Remove a product from the inventory.
   * 4. Update Product: Modify the quantity or price of an existing product.
   * 5. Calculate Total Value: Calculate and display the total monetary value of all items in the inventory.
   * 6. Exit: Terminate the program.
3. **Input Validation (try-except):**
   * When adding or updating quantities/prices, ensure that the input is numeric (integer for quantity, float for price). Use try-except blocks to catch ValueError if non-numeric input is provided.
   * If a product is to be removed or updated, check if it exists in the inventory. If not, inform the user.
   * Ensure quantities are non-negative.
4. **User-Friendly Messages:** Provide clear and informative messages to the user for successful operations, errors, and invalid choices.