# Case Studies

Instructions: Complete both of the case studies in this document. Be sure to document your code so that some who isn’t familiar with the code can follow along. For both programs, insert a screenshot of the console from the program into this document. Also include screenshots of the program handling errors such as file not found, invalid integer, invalid float, etc. Submit your python code to Blackboard. If you are submitting more than one file per case study, zip the files and submit the zip file. You can use IDLE or VS Code.

These case studies take the place of a mid term exam. They are open book, open notes, open Google, but you may NOT collaborate with classmates on the solutions. By submitting this document to Blackboard, you are stating that the work contained in the file and in the programs is your own.

## Case Study 1: Data Importer

Create a program that imports sales data from one or more CSV files and allows the user to add sales data. Include error handling to check for invalid data entered into the program. The menu should look like this:

Table

Description automatically generated with medium confidence

The entry screen should look like this:

Table

Description automatically generated

Specifications

* The program should accept float entries for the amount and integer entries for the year, month, and day.
* Store the functions for getting sales data from a user in a module named sales. Use docstrings and type hints to document the sales module.
* Calculate the quarter based on the month value:
  + 1, 2, 3: Quarter 1
  + 4, 5, 6: Quarter 2
  + 7, 8, 9: Quarter 3
  + 10, 11, 12: Quarter 4
* Use a list to store the sales data for each user entry, and use a list of lists to store multiple entries.
* Create a menu that allows the user to view existing sales data, add new sales data, and import sales data.
* Allow users to import sales data from a CSV file that contains sales amount, year, month, and day. Use a text file to keep track of imported files so a user can’t import a file more than once.
* Store the functions for writing and reading the files in a separate module named db.py.
* When the program starts, it should read the sales data from a file named all\_sales.csv.
* When the program ends, it should write the sales data, including any imported data and any data entered by the user, to all\_sales.csv.
* Don’t store the quarter value. Instead, calculate it from the month as needed.
* Handle the exception that occurs if the program can’t find the all\_sales.csv file, the file the user is trying to import, or the text file that tracks imported files. Handle the exceptions that occur if the user enters a string or float where an integer is expected, or a string where a float is expected.
* Handle the exceptions that occur if the data in the imported file can’t be converted to a float or an int. Use asterisks and question marks to notify the user of bad imported data.
* The program should validate the user’s entries as follows:
  + The sales amount should be greater than zero.
  + The sales month should be between 1 and 12.
  + The sales day should be between 1 and 31, except:
    - For months 4, 6, 9, and 11, the maximum day value is 30.
    - For month 2, the maximum day value is 28 (don’t handle leap years).
  + The sales year should be greater than or equal to 2000 and less than or equal to 9999.

Graphical user interface, text, email

Description automatically generated

## Case Study 2 – Baseball Team Management

Create a program that lets the manager of a baseball team keep the data for each player and list and display the lineup for a baseball game. Include error handling to check for invalid data entered into the program. The menu should look like this:

Graphical user interface, text, email

Description automatically generated

The display lineup screen should look like this:

Table

Description automatically generated

**Specifications**

* The formula for calculating batting average is: **average = hits / at\_bats**
* The program should round batting average to a maximum of three decimal places.
* Use functions to organize the code to make it more reusable, easier to read, and easier to maintain.
* If the user enters an invalid menu option, display an error message and display the menu again so the user can clearly see the valid menu options.
* Make sure the user can’t enter data that doesn’t make sense (such as a negative number of hits or the player having more hits than at bats).
* Use a list of lists to store each player in the lineup.
* Use a tuple to store all valid positions (P, C, 1B, 2B, 3B, SS, LR, CF, RF).
* When entering/editing positions, the program should always require the user to enter a valid position.
* Use a CSV file named players.csv to store the lineup.
* Store the functions for writing and reading the file of players in a separate module named db.py.
* Handle the exception that occurs if the program can’t find the data file.
* Handle the exceptions that occur if the user enters a string where an integer is expected.
* Handle the exception that occurs if the user enters zero for the number of at bats. In that case, the player’s batting average should be 0.0.

Table

Description automatically generated

Errors:

A picture containing application

Description automatically generated

A picture containing text

Description automatically generated

