COMP218 Assignment4

# AUID	3608550
Deadline	@January 7, 2026
StudentName	Boris Bojanov
files & media	Assignment4.pdf

https://docs.python.org/3/library/tkinter.html#tkinter-modules

For this last assignment, you are required to complete one of the three projects listed below. Complete and submit it after you have finished working through Unit 10.
Your submission must include the following files:
• (10 marks) Your assignment report
• (5 marks) The program files you developed for the project you chose for this assignment, the
Jupyter Notebook files you created and used during your study of Unit 10, and the Jupyter
Notebook you created and used when working on this assignment

Format :

Your assignment report should be in either (a) Markdown format (.md), written in VS Code, or (b) Word format. When you create a .md file in VS Code, VS Code recognizes it as a Markdown document and you can then use all the Markdown tags to format the content. For Assignment 4, name the report assignment-report.md or assignment-report.docx. Place it in a subfolder called assignment4 within your main folder called comp218. The assignment report must begin with a cover page stating the course number and name, the assignment number, your name and student ID, and your best estimate of your time spent on the assignment. The cover page will then be followed by your documentation for work on the project.

Your documentation should include

1. your interpretation of the project;
2. an analysis of the tasks and requirements, techniques, and/or algorithms you used to solve the
   problems or accomplish the tasks;
3. an explanation of modules, classes, and functions you wrote and/or used in the system developed
   for the project; and
4. a brief user guide.

When you submit the assignment, all the files for the assignment, including the Jupyter Notebook files,
must reside in the folder named assignment4. Compress the entire assignment4 folder into a file named
assignment4.zip and submit the zip file to the Assignment 4 drop box on the course home page.

Project 1: GUI-Based Course Management System (85 marks total)

Study relevant materials available in the course or on the internet and in VS Code when answering the following questions.

1. (10 marks) Define a student class modelling a student, including name, student ID, start date in the class, name of tutor assigned, and a list of assessment records. Each assessment record will be a tuple containing a number to identify the assessment, the weight of the assessment, and the mark, which should be zero for a newly added instance to a course.

```
from typing import List



class Student:

    def __init__(self, name, student_id, start_date, tutor_name, assessments:
List[tuple] = None):
        if assessments is None:
            assessments = list() # Initialize to empty list if not provided
        self.name = name
        self.student_id = student_id
        self.start_date = start_date
        self.tutor_name = tutor_name
        self.assessments = assessments
```

2. (10 marks) Define a course class modelling a course, including the course number such as
comp218, title, revision number, date of initial offering, and list of students, and assessment schedule as a list of assessment items. Each assessment item is represented as a tuple of (ID, name, weight) in which the ID is the assessment item ID; name is the assessment name, such as Assignment 1; and final exam. Weight is the percentage of the assessment towards the final grade.

```
class Course:

    def __init__(self, name, nameAbreviated, title, revisionNumber, dateOffer
ed, studens:List[Student]=[], professorName='', assessments:List[tuple] =
None ):
        if assessments is None:
            assessments = list() # Initialize to empty list if not provided
        self.name = name
        self.nameAbreviated = nameAbreviated
        self.title = title
        self.revisionNumber = revisionNumber
        self.date = dateOffered
        self.registeredStrunds = studens
        self.professorName = professorName
        self.assessments = assessments
```

3. (55 marks) The GUI application should meet the following criteria:

   a. There should be a button for adding a new course to the system, which will open a form for input and save basic course info previously mentioned, with the list of students empty. Note that when saving the basic course info, the system should be able to check whether the total weight of all assessment items make up 100%.

      Done

   b. When a new course is added to the system, a unique binary file will be created for permanent storage of the course data.

      Done

   c. At the start of the system, the system should automatically load all courses from their respective binary files to restore their internal object representation.

      Done

   d. There should be a button to get a list of courses being offered to students.

      Done

   e. The user should be able to select a course from the list.

      Done

   f. The user should be able to add students to the selected course.

      chose from saved list of student OR create a new student

   g. The user should be to see a list of students in the course.

      Done

h. The user should be able to select a student from the list.

   Done

i. The user should be able to record an assessment for the selected student.

   Done

j. The system should automatically calculate display the final grade of the student for the course.

   Done

k. The user should be able to see a list of assessments, including the calculated final grade for the selected student.

   Done

l. There should be a button to shut down the system, but before shutting down the application, the system must save/pickle the data for each course back to its binary file.

   Done

4. (5 marks) Your analysis and design of the system should be well documented in your assignment report.

   Done

5. (5 marks) Within each of your program files, there should be a docstring at the beginning stating the name and purpose of the file, as well as the ownership and revision history. One docstring is required for each class and function/method class. End-of-line comments are desired when deemed necessary.

   Done

The Project

This project is to build a small desktop **Course Management System** using Python and Tkinter that lets a user create and manage course data across multiple program runs. The system must support two main “entities”:

- **Courses** (basic course info + assessment structure + a list of enrolled students)
- **Students** (basic student info + marks for each assessment item in the course)

A key requirement is **persistence**: when the program is closed and reopened, the data must still be there. To meet that, each course is stored in its **own binary file** (pickled object). When the program starts, it automatically loads all `.bin` course files from a data folder and rebuilds the internal list of course objects.

Functionally, the GUI is expected to guide the user through a workflow:

a. Create a course and define its assessments (weights must total 100%).
b. View the list of courses and select one course to work with.
c. Add students to the selected course.
d. View and select a student from the selected course.
e. Record assessment marks for the selected student.
f. Automatically compute and display the student's final grade.
g. View an assessment breakdown and the computed final grade.
h. Shut down safely and ensure every course is saved back to its binary file before exit.

Overall, the system is an object-based GUI application that combines:

The Tkinter library for the UI screens AND windows.

The client side input validation and enforcement of basic business rules.

Selection handling for course/student/assessment selection.

The pickle library for pickling/unpickling the `.bin` files which allow for storage of data that cannot be read directly like plain text

---

Analysis of Tasks and Requirements

Requirements Breakdown:

1. Create a course and define its assessments (weights must total 100%).
2. View the list of courses and select one course to work with.
3. Add students to the selected course.
4. View and select a student from the selected course.
5. Record assessment marks for the selected student.
6. Automatically compute and display the student's final grade.
7. View an assessment breakdown and the computed final grade.
8. Shut down safely and ensure every course is saved back to its binary file before exit.

A. Core data model and persistence of data

Problem: The program must store course data permanently and restore it at startup.

Technique used:

- Store each `Course` object as a **pickle** (`.bin`) file.
- Use a `courseData/` directory relative to the script location.
- On startup, scan that directory for `.bin` files and unpickle them to rebuild `self.courses`.

Why this works well:

- Pickle allows storing entire Python objects (including nested lists like students and assessments).
- Each course having its own file matches the assignment requirement ("unique binary file per course").
- Loading all files at startup ensures continuity between sessions.

Implementation details:

- `checkDataDirectory()` ensures the folder exists.

- `saveCourseToFile(courseObj)` generates a safe filename and pickles the course.
- `loadCourses()` unpickles all course files.
- To support updates, each loaded course stores `courseObj.filepath`, so changes can be saved back to the same file later (important for students/marks updates).

B. GUI structure, Tkinter frames and windows

Problem: Provide a usable interface that can create courses, pick a course/student, and enter marks.

Techniques used:

- Tkinter `Tk()` as the main window, and `Toplevel()` windows for secondary forms (course form, course list, student list, assessment form, etc.).
- Layout is done primarily using the **grid geometry manager** to keep the UI structured and readable.
- A consistent "widget factory" pattern (`createButton`, `createLabel`, `createEntry`, `createListbox`) reduces repetitive code and keeps layout styles consistent.

Why grid + helper functions help:

- Grid makes forms (label + entry) easy to align in rows/columns.
- Helper functions enforce consistent usage and avoid mistakes like passing grid options into widget constructors.

C. Input validation and business rules

This project requires several **constraints** and checks to prevent invalid data from being stored.

1. Assessment weights must total 100%

A course can only be submitted if assessments sum to exactly 100%.

Technique used:

- Maintain a list of assessments `(id, name, weight)`.
- Maintain a running total (`self.totalWeight`).

- Disable the “Submit Course” button until the total is 100%.
- Prevent adding an assessment if it would push total above 100.

Algorithm:

- `total = sum(weight for each assessment)`
- If total < 100 → allow adding assessments, block submit
- If total == 100 → block adding assessments, allow submit
- If total > 100 → show error (prevented earlier)

This is a effective constraint-checking pattern for the expected form based entries.

2. Valid assessment weights and marks

- Weight must be a positive integer.
- Recorded marks must be numeric and between 0–100.
- Try/except for numeric conversion.
- Range checks with error messages via `messagebox.showerror()`.

3) Prevent duplicate student IDs, within a course

Requirement (practical): Student IDs should be unique inside a course to avoid ambiguous records.

Technique used:

- Before appending a new student, scan existing students and compare IDs.

D. Selection handling with `Listbox`

Problem: The program must let the user select a course and a student from lists, then act on those selections.

Technique used:

- Tkinter `Listbox` displays human-readable strings.
- The program stores the “real” objects in parallel lists:

    - `self.courses` is the source list for the course list-box.
    - `selectedCourse.registeredStudents` is the source list for the student list box.
    - `selectedStudent.assessments` is the source list for the assessment list box.
- When the user clicks an item, `listbox.curselection()` returns the selected row index (0-based).
- That index is used to fetch the matching object/tuple from the underlying list.

List boxes store strings, not objects.

- Using the row index as a "pointer" into the underlying Python list is the standard way to map UI selection to real data.

User experience improvement:

- The "Select" button stays disabled until the user actually picks an item (`<<ListboxSelect>>` event).
- Double-click triggers selection immediately.

E. Recording marks and final grades

Requirement:

- (i) Record an assessment mark for the selected student.
- (j) Automatically calculate and display final grade.
- (k) Display a list of assessments including the final grade.

Data representation choices:

- Course assessments are stored as: `(assessment_id, assessment_name, weight)`
- Student assessments are stored as: `(assessment_id, weight, mark)`

This makes grade calculation straightforward because each student assessment tuple contains its own weight and current mark.

Algorithm for final grade (weighted average):

For each assessment item:

- convert mark percent to fraction: `mark / 100`

- multiply by its weight: `(mark/100) * weight`
- sum all contributions

So:

$$\text{Mean Final Grade} = \sum_{i}^{n} (\bar{y}_i * \text{weight}_i)$$

$$= \sum_{i}^{n} \frac{\text{mark}_i}{100} * \text{weight}_i$$

- Update the selected assessment tuple by replacing it (tuples are immutable).
- Save the updated course back to its file immediately after recording marks (prevents data loss).
- Provide a "View Assessments + Final Grade" window that lists each assessment and its computed contribution, plus the final grade at the top.

F. Safe shutdown (save n exit)

Requirement (I): Before shutting down, the system must save/pickle each course back to its binary file.

Provide a `shutdown()` function that:

1. confirms exit
2. iterates through all courses in `self.courses`
3. writes each course back to its `filepath` (or creates one if missing)
4. then closes the app

Why this matters:

- In GUI apps, users may close the window without saving.
- A central shutdown routine guarantees persistence and prevents partial data loss.

Extra reliability step:

- Bind the window close button (the “X”) to the same shutdown routine using `WM_DELETE_WINDOW`.

Modules, Classes, and Functions

The Course Management System is implemented using a **modular, object-oriented design**. The system is divided into three main Python modules, each with a well-defined responsibility: data representation, student representation, and graphical user interaction. This separation improves readability, maintainability, and extensibility of the system.

3.1 Module Overview

3.1.1 CourseClass.py

This module defines the **Course** data structure used throughout the system. A course represents an academic offering and contains both administrative information and academic structure, including enrolled students and assessment definitions.

The module encapsulates all attributes related to a course, making it suitable for serialization (pickling) and long-term storage.

CourseClass

3.1.1 StundentClass.py

This module defines the **Student** class, which represents an individual student enrolled in a course. Each student object stores personal information and a list of assessment results that correspond to the course’s assessment structure.

By isolating student logic in its own module, the system cleanly separates student-level data from course-level data.

StundentClass

3.1.3 CourseManagmentGUI.py

This module implements the **graphical user interface (GUI)** using the Tkinter library. It serves as the controller of the application, coordinating user interaction, data validation, object creation, persistence, and workflow logic.

The GUI module connects user actions (button clicks, list selections) to underlying course and student objects, enforcing the project's functional requirements.

CourseManagmentGUI

3.2 Class Descriptions

3.2.1 Course Class

The `Course` class models a single academic course.

Attributes:

- `name`: Full course name (e.g., *Computer Science 482: Human-Computer Interaction*)
- `nameAbreviated`: Short identifier (e.g., *COMP 482*)
- `title`: Course title
- `revisionNumber`: Course revision identifier
- `date`: Initial offering date
- `professorName`: Instructor's name
- `registeredStudents`: List of `Student` objects enrolled in the course
- `assessments`: List of assessment definitions as tuples `(ID, name, weight)`
- `filepath`: Path to the binary file used for persistence

Design goals:

- The class is intentionally lightweight and data-focused.
- Lists for students and assessments are initialized carefully to avoid shared mutable defaults.
- The class is fully pickle-compatible, allowing direct serialization to disk.

This class acts as the **core persistent unit** of the system.

3.2.2 `Student` Class (`StundentClass.py`)

The `Student` class represents a student enrolled in a specific course.

Key attributes:

- `name` : Student's full name
- `studentID` : Unique student identifier
- `startDate` : Enrollment start date
- `tutorName` : Assigned tutor
- `assessments` : List of tuples `(assessmentID, weight, mark)`

Important methods:

- `addAssessment()` : Adds an assessment record to the student
- Getter and setter methods for name and assessments
- `__str__` / `__repr__` : Provide readable string representations for debugging and display

Design considerations:

- Each student stores their own copy of assessment weights and marks, enabling independent grade calculation.
- Newly added students start with assessment marks initialized to zero.

3.2.3 `CourseManagementGUI` Class (`CourseManagmentGUI.py`)

The `CourseManagementGUI` class controls the entire application workflow and user interface.

This class is responsible for:

- Initializing the GUI
- Loading and saving course data
- Managing user selections
- Enforcing business rules
- Coordinating interactions between `Course` and `Student` objects

3.3 Key Functions and Responsibilities

3.3.1 GUI Helper Functions

To reduce repetition and enforce consistent layout, several helper functions are used:

- `createButton()`
- `createLabel()`
- `createEntry()`
- `createListbox()`
- `createFormEntry()`

These functions standardize widget creation and ensure all UI elements use the `grid` layout consistently.

3.3.2 File Handling and Persistence Functions

- `checkDataDirectory()` : Ensures the course data directory exists
- `safeafyFileName()` : Sanitizes strings for safe filenames
- `courseFilename()` : Generates unique filenames per course
- `saveCourseToFile()` : Serializes a new course to disk
- `loadCourses()` : Loads all existing course files at startup
- `saveSelectedCourse()` : Updates the currently selected course file
- `saveAllCourses()` : Saves **all courses** before shutdown

These functions collectively ensure that **no user data is lost between sessions**.

3.3.3 Course Management Functions

- `newCourseForm()` : Opens the course creation form
- `addAssessment()` : Adds assessment items while enforcing a 100% total weight rule
- `computeTotal()` : Tracks assessment weight totals and controls form submission

- `submitNewCourse()`: Creates and saves a fully validated course
- `showCourseList()`: Displays all available courses and allows selection

3.3.4 Student Management Functions

- `addStudentForm()`: Adds a new student to the selected course
- `showStudentList()`: Displays and selects students from a course
- Duplicate student IDs are prevented within a course

3.3.5 Assessment and Grading Functions

- `recordAssessmentForm()`: Records marks for a selected student's assessment
- `calculateFinalGrade()`: Computes the weighted final grade automatically
- `showStudentAssessments()`: Displays all assessments with individual contributions and final grade

The final grade is calculated using a weighted average formula.

3.3.6 Shutdown Handling

- `shutDown()`: Handles safe program termination
  - Prompts user confirmation
  - Saves all courses to their binary files
  - Closes the application cleanly
- The same function is bound to both the **Quit button** and the window close event (`WM_DELETE_WINDOW`)

This ensures compliance with the assignment requirement that all data is saved before exit.

User Guide

A brief guide on how to use the Course Management System application. The system is menu-driven and uses buttons and list selections to guide the user through all required tasks.

Starting the Application

1. Run the program by executing `CourseManagmentGUI.py`.
2. When the application starts:
    - The main window appears.
    - All previously saved courses are automatically loaded from the `courseData` directory. This is where the `.bin` files are stored.
    - The user can immediately begin working with existing courses or create new ones.

Creating a New Course

1. From the main window, click **"Add New Course"**.
2. Enter the required course information:
    - Course Name
    - Course Abbreviated Name
    - Course Title
    - Revision Number
    - Date Offered
    - Professor Name
3. Add assessment items:
    - Enter an assessment name and weight.
    - Click **"Add Assessment"**.
    - Repeat until the total assessment weight equals **100%**.
4. When the total reaches 100%, the **"Submit Course"** button becomes enabled.
5. Click **"Submit Course"** to save the course.

- The course is stored as a binary file and becomes available in the course list.

Viewing and Selecting a Course

1. From the main menu, click **“Show List of Courses”**.
2. A window displays all available courses.
3. Click on a course in the list to select it.
4. Click **“Select”** (or double-click the course).
5. The selected course becomes the active course for student and assessment operations.

Adding Students to a Course

1. Ensure a course is selected.
2. Click **“List Students (Selected Course)”**.
3. In the student window, click **“Add Student to Selected Course”**.
4. Enter the student’s:
   - Name
   - Student ID
   - Start Date
   - Tutor Name
5. Click **“Add Student”**.
   - The student is added to the selected course.
   - Duplicate student IDs within the same course are not allowed.

Viewing and Selecting Students

1. With a course selected, click **“List Students (Selected Course)”**.
2. A list of enrolled students is displayed.

3. Click a student to select them.
4. Click **“Select Student”** (or double-click the student).
5. The selected student becomes the active student for assessment entry and grade viewing.

Recording Assessment Marks

1. Ensure both a course and a student are selected.
2. Click **“Record Assessment (Selected Student)”**.
3. Select an assessment from the list.
4. Enter the student’s mark (0–100).
5. Click **“Save Mark”**.
    - The assessment mark is stored.
    - The student’s final grade is automatically recalculated and displayed.

Viewing Assessments and Final Grade

1. Ensure a student is selected.
2. Click **“View Assessments + Final Grade”**.
3. A window displays:
    - Each assessment
    - Assessment weight
    - Recorded mark
    - Contribution to final grade
    - Overall final grade for the course

Exiting the Application Safely

1. Click **“Quit”** or close the window using the window close button.
2. The system prompts for confirmation.

3. Upon confirmation:
    - All courses are saved back to their binary files.
    - The application shuts down safely.

This ensures that all entered data is preserved for future sessions.