MacEwan UNIVERSITY

Project

General information

Python/IDE Python 3/Wing IDE 101

	Milestone #1	**Milestone #2**	**Milestone #3**
Lab time	Feb. 11-15	Mar. 11-15	Mar. 25-29
Due date	Feb. 22 (Friday)	Mar. 22 (Friday)	Apr. 5 (Friday)
Due time	11:59:59 PM	11:59:59 PM	11:59:59 PM
Weight	7%	7%	6%

Topics

- ✓ Text/mutability
- ✓ File input
- ✓ Lists
- ✓ Dictionaries/sets
- ✓ Data encoding
- ✓ `graphics.py`
- ✓ Algorithms

Submission

- ✓ **The code file (`project`*`initials`*`.py`) should be submitted electronically** according to the instructions from your lab instructor.
- ✓ A portion of the total marks will be allocated to programming style. For example, functions should be reasonably small, names should be meaningful and descriptive, naming conventions should be followed consistently, code should be formatted properly, and comments should be accurate and well written.
- ✓ Comments are **required** for:
  - The program indicating the student name and program name.
  - EACH function indicating the function purpose, parameters, and return value.
  - Any block of code for which the purpose may be unclear. Note: you should always try to write code that can be understood easily without comments.

Introduction

You will develop a Python program that implements a gradebook. It will allow a student to track course performance, calculate grade point averages, and plot scores over time.

The following paragraphs provide an overview of each milestone.

Milestone #1 focuses on (a) developing a data structure to store courses, activities, and scores; (b) reading courses, activities, and scores from the user; (c) removing courses from the gradebook; and (d) displaying the gradebook for the user.

Milestone #2 will likely involve (a) adding error handling, (b) saving/loading the gradebook data, and (c) displaying GPAs.

Milestone #3 will likely involve (a) creating a graphical window, (b) displaying a plot of scores over time, and (c) interactively displaying course information.

General requirements

Regardless of the milestone, your program must meet the following general requirements:

1. Your program must not run when imported as a module, e.g., imported by eSubmit. To prevent eSubmit from running your program, do not call the `main` function in your code.
2. Your program must only terminate when the user selects the menu's "Quit" option.
3. For milestones #2 and #3, your program must handle all reasonable user errors – not just the errors described explicitly in this specification. For example, the following "problems" should not cause the program to crash:
   - attempting to open a file that doesn't exist (Milestone #2);
   - providing unexpected input types, e.g., alphabetic characters when a string of digits is expected (Milestone #2); and
   - closing the plot (Milestone #3).

   You may want to use `try` and `except` at critical places in your code.

Milestone #1: Gradebook data (due Feb. 22)

The tasks in this milestone relate to obtaining, manipulating, and displaying gradebook data. As the program obtains data from the user, it must store these data within an appropriate data structure. Prior to writing any code, sketch (on paper) an appropriate data structure that uses dictionaries and/or lists.

When the user calls a function named `main`, the program displays the following menu:

```
===␣Gradebook␣===
(1)␣Add␣course
(2)␣Add␣score
(3)␣Remove␣course
(4)␣Display␣gradebook

(5)␣Reserved␣for␣future␣use
(6)␣Reserved␣for␣future␣use

(7)␣Reserved␣for␣future␣use

(0)␣Quit
```

Observe the one blank line before and one blank line after the menu.

After displaying the menu, the program displays the following prompt:

```
Enter␣command:␣
```

For Milestone #1, only menu options 1, 2, 3, 4, and 0 work. If the user selects any other option, the program simply redisplays the prompt (without inserting a blank line).

For example:

```
Enter␣command:␣
Enter␣command:␣add
Enter␣command:␣5
Enter␣command:␣7
Enter␣command:␣-123
Enter␣command:␣1

Course␣name:
```

Add course

When the user selects menu option 1, the program prompts for the information that it needs to add a course to the gradebook. It prompts for a course name and then repeatedly prompts for activities and weights. As soon as the user enters an empty string for an activity, the program stops obtaining activities.

Consider the following example of the expected output:

```
…
Enter␣command:␣1

Course␣name:␣CMPT␣103
Evaluation:
␣␣Activity:␣Labs
␣␣Weight:␣20
␣␣Activity:␣Project
␣␣Weight:␣15
␣␣Activity:␣Midterm␣#1
␣␣Weight:␣15
␣␣Activity:␣Midterm␣#2
␣␣Weight:␣15
␣␣Activity:␣Final
␣␣Weight:␣35
␣␣Activity:␣

===␣Gradebook␣===
…
```

Additional requirements:

- the weights are floating-point numbers
- the weights do not need to sum to 100
- if the course name is an empty string or it already exists in the gradebook, the program displays an error message and continues to prompt for a valid course name, i.e.,

  ```
  Course␣name:␣CMPT␣103
  Error:␣the␣course␣name␣already␣exists
  Course␣name:␣
  Error:␣the␣course␣name␣cannot␣be␣empty
  Course␣name:␣CMPT␣101
  Evaluation:
  ␣␣Activity:␣

  ===␣Gradebook␣===
  …
  ```

- if the activity name already exists for the course, the program similarly displays an error message: `Error:␣the␣activity␣name␣already␣exists`

Add score

When the user selects menu option 2, the program prompts for the information that it needs to add a score to the gradebook.

It first prompts for the course. Entering a `?` or a non-existent course name produces a list of available courses (displayed in sorted order). Entering the empty string aborts the process and causes the program to return to the menu.

If the user enters a valid course, the program prompts for the activity. Entering a `?` or a non-existent activity produces a list of available activities (displayed in sorted order). Entering the empty string aborts the process and causes the program to return to the menu.

If the user enters a valid activity, the program prompts for the completion date, i.e., when it was submitted, and finally, it prompts for the score. The former should be stored as a `date` object from Python's `datetime` library. The latter should be a floating-point number.

After the user enters the date and score, the program updates the gradebook and returns to the main menu.

Consider the following example of the expected output:

```
…
Enter␣command:␣2

Course␣(?␣for␣list):␣?
␣␣Courses:␣CMPT␣103
Course␣(?␣for␣list):␣invalid␣name
␣␣Courses:␣CMPT␣103
Course␣(?␣for␣list):␣CMPT␣103
Activity␣(?␣for␣list):␣?
␣␣Activities:␣Final,␣Labs,␣Midterm␣#1,␣Midterm␣#2,␣Project
Activity␣(?␣for␣list):␣invalid␣name
␣␣Activities:␣Final,␣Labs,␣Midterm␣#1,␣Midterm␣#2,␣Project
Activity␣(?␣for␣list):␣Labs
Completed␣date␣(YYYY-MM-DD):␣2019-01-31
Score␣(%):␣90

===␣Gradebook␣===
…
```

Remove course

When the user selects menu option 3, the program removes a course from the gradebook. The program prompts for a course exactly like when adding a score.

Consider the following example of the expected output:

```
…
Enter␣command:␣3

Course␣(?␣for␣list):␣does␣not␣exit
␣␣Courses:␣CMPT␣103
Course␣(?␣for␣list):␣?
␣␣Courses:␣CMPT␣103
Course␣(?␣for␣list):␣CMPT␣103

===␣Gradebook␣===
…
```

Additional requirements:

- entering the empty string aborts the process and results in ab lank line followed by the menu

Display gradebook

When the user selects menu option 4, the program displays the course, activity, and score data entered into the gradebook. The *format* of this output must match the following example:

```
…
Enter␣command:␣4

Course:␣CMPT␣103
␣␣Activity:␣Final␣(Weight:␣35.0)
␣␣Activity:␣Labs␣(Weight:␣20.0)
␣␣␣␣Score:␣90.0␣(Completed:␣2019-01-31)
␣␣Activity:␣Midterm␣#1␣(Weight:␣15.0)
␣␣Activity:␣Midterm␣#2␣(Weight:␣15.0)
␣␣Activity:␣Project␣(Weight:␣15.0)

===␣Gradebook␣===
…
```

Additional requirements:

- the courses must appear in sorted order
- the activities must appear in sorted order
- if a score exists for an activity, the program displays the score as a floating-point number

Quit

When the user selects menu option 0, the program immediately terminates.

Milestone #2 (due Mar. 22)

The tasks in this milestone relate to (a) improving your program's error handling when obtaining user input, (b) writing/reading data structures to/from files, and (c) using stored data in calculations. You will improve the error handling related to user input using `try`/`except` to catch exceptions. You will add code to the function `main` to read your gradebook data structure from a file and write it to a file. Finally, you will write two functions that use gradebook data for some useful calculations.

> **Solutions should not use any global variables.**
> Create your gradebook data structure inside the `main` function. Pass that data structure (or parts of it) to other functions as necessary.

Improve error handling

In all places where your program reads integer numbers, floating-point numbers, and dates, improve the error handling. Use Python's `try`/`except` functionality around the related conversions. If the user provides a value that cannot be converted to a number/date, display a one-line error message and redisplay the prompt (rather than crash).

Expected error messages:

- For bad weights:

```
␣␣Error:␣the␣provided␣weight␣is␣invalid
```

- For bad dates:

```
Error:␣the␣provided␣date␣is␣invalid
```

- For bad scores:

```
Error:␣the␣provided␣score␣is␣invalid
```

Example:

```
…
Course␣(?␣for␣list):␣CMPT␣103
Activity␣(?␣for␣list):␣Labs
Completed␣date␣(YYYY-MM-DD):␣2019-01-41
Error:␣the␣provided␣date␣is␣invalid
Completed␣date␣(YYYY-MM-DD):␣2019-01-01
Score␣(%):␣fifty
Error:␣the␣provided␣score␣is␣invalid
Score␣(%):␣50

===␣Gradebook␣===
…
```

Write/read data structures to/from files

Write to a file

When the program terminates, it must save the gradebook data structure (even if empty) to a binary file named `gradebook.p` and display whether it succeeded or failed. If it fails to write the file, it must not crash. See the examples below for the expected output.

Expected output when it succeeds:

```
…
(0) Quit

Enter command: 0

Data saved in gradebook.p
```

Expected output when it fails:

```
…
(0) Quit

Enter command: 0

Failed to save data in gradebook.p
```

Read from a file

When the user calls the function `main`, the program must read the gradebook data structure from the binary file and display whether it succeeded or failed. If it fails to read the file, it must start with an empty gradebook (as was done in milestone #1) and create the file when the program terminates.

Expected output when it succeeds:

```
>>> main()
Data loaded from gradebook.p

=== Gradebook ===
(1) Add course
…
```

Expected output when it fails:

```
>>> main()
Failed to load data from gradebook.p

=== Gradebook ===
(1) Add course
…
```

Calculate possible outcomes

Update menu option 5 to read

```
(5) Calculate possible outcomes
```

This menu option calculates possible outcomes for a course while accounting for the scores entered in the gradebook. It prompts for a course in the exact same way as when adding a score. If the user provides an empty string as the course name, it returns to the menu. If the user provides a valid course name, it produces three lines of output that correspond to three scenarios:

1. the worst case, which is based on the entered scores (when available) and scores of 0% on all remaining activities,
2. the expected case, which is based on the weighted average of the entered scores (when available) and the assumption that performance will be consistent for the remaining activities, and
3. the best case, which is based on the entered scores (when available) and scores of 100% on all remaining activities.

You may assume that the weight of a course's activities will always sum to 100.

Important: use Python's f-strings to format floating-point numbers to 1 decimal place. Python will round your floating-point numbers.

Formatted string literals

Python 3.6 introduced a new feature named the f-string. This feature provides the easiest option for creating strings where your floating-point numbers are rounded to 1 decimal place. The documentation is available at:

- `https://docs.python.org/3/reference/lexical_analysis.html#formatted-string-literals`

Consider the following examples:

```
>>> name␣=␣"Bob"
>>> print(f"Hello␣{name}.")
Hello␣Bob.
>>> num␣=␣123.4567
>>> print(f"Value:␣{num:.3f}.")
Value:␣123.457.
>>> print(f"Value:␣{num:.1f}.")
Value:␣123.5.
>>>
```

To convert percentages to letter grades, use the grading scheme provided in your CMPT 103 course outline. For example, a percentage greater than or equal to 98.0 is an A+.

Expected output (Example #1):

```
Enter␣command:␣4

Course:␣CMPT␣123
␣␣Activity:␣Final␣(Weight:␣60.0)
␣␣Activity:␣Midterm␣(Weight:␣40.0)
␣␣␣␣Score:␣56.75␣(Completed:␣2019-01-01)

===␣Gradebook␣===
(1)␣Add␣course
…
(0)␣Quit

Enter␣command:␣5

Course␣(?␣for␣list):␣CMPT␣123
␣␣Worst␣case:␣␣␣␣22.7␣(F)
␣␣Expected␣case:␣56.8␣(C-)
␣␣Best␣case:␣␣␣␣␣82.7␣(B+)

===␣Gradebook␣===
…
```

Expected output (Example #2):

```
Enter␣command:␣4

Course:␣CMPT␣234
␣␣Activity:␣Activity␣1␣(Weight:␣25.0)
␣␣␣␣Score:␣40.0␣(Completed:␣2019-01-01)
␣␣Activity:␣Activity␣2␣(Weight:␣25.0)
␣␣␣␣Score:␣80.0␣(Completed:␣2019-01-01)
␣␣Activity:␣Activity␣3␣(Weight:␣25.0)
␣␣Activity:␣Activity␣4␣(Weight:␣25.0)

===␣Gradebook␣===
(1)␣Add␣course
…
(0)␣Quit

Enter␣command:␣5

Course␣(?␣for␣list):␣CMPT␣234
␣␣Worst␣case:␣␣␣␣30.0␣(F)
␣␣Expected␣case:␣60.0␣(C)
␣␣Best␣case:␣␣␣␣␣80.0␣(B+)

===␣Gradebook␣===
…
```

Calculate GPA

Update menu option 6 to read

```
(6) Calculate GPA
```

This menu option calculates the average GPA based on all of the courses/scores in the gradebook. For each course, it calculates a weighted average based on all activities with scores (the "expected case" from the previous part). From that average, it obtains the corresponding GPA for the course. Finally, it averages all of those GPAs to display the overall GPA. Display the overall GPA to one decimal place.

To convert percentages to GPAs, use the grading scheme provided in your CMPT 103 course outline.

Example output:

```
Enter␣command:␣4

Course:␣CMPT␣101
␣␣Activity:␣Coursework␣(Weight:␣100.0)
␣␣␣␣Score:␣72.0␣(Completed:␣2019-01-01)
Course:␣CMPT␣102
␣␣Activity:␣Coursework␣(Weight:␣100.0)
␣␣␣␣Score:␣77.0␣(Completed:␣2019-01-01)
Course:␣CMPT␣103
␣␣Activity:␣Coursework␣(Weight:␣100.0)
␣␣␣␣Score:␣87.0␣(Completed:␣2019-01-01)

===␣Gradebook␣===
(1)␣Add␣course
…
(0)␣Quit

Enter␣command:␣6

Expected␣GPA:␣3.1

===␣Gradebook␣===
(1)␣Add␣course
…
```

Milestone #3 (due Apr. 5)

Important: multiple versions of `graphics.py` exist. For this milestone, you must use the latest version: "Version 5 8/26/2016". The version of `graphics.py` can be found within the file itself.

The tasks in this milestone relate to (a) extending the menu and verifying that sufficient data exist for plotting, (b) creating a window, a rectangle (to serve as the plot region), and labelled axes, (c) plotting a course's scores within the rectangle, and (d) recognizing and responding to mouse clicks within the window.

Extending the menu and verifying sufficient data

Update menu option 7 to read

```
(7) Plot scores
```

This menu option prompts for a course in the exact same way as when adding a course. If the user provides an empty string as the course name, it returns to the menu. If the user provides a valid course name, it proceeds to validate whether sufficient data exist for plotting.

Expected error messages:

- If the course does not have any graded activities:

```
Error: course does not have graded activities
```

- If the graded activities all occur on the same day:

```
Error: graded activities all occur on the same day
```

In the case of an error, the program returns to the menu. Given no errors, the program proceeds to plot scores.

Plotting

Given data to plot, the program proceeds to create a window with the following properties:

- dimensions: 1024×600 pixels
- title: the course name

In the window, the program must display a rectangular plot region using a rectangle. The rectangle's dimensions are always 824×400 pixels. The plot region must be centred in the window with 100 pixels of padding on each of the four sides – regardless of the plot's domain.[1] The left edge of the rectangular plot region must coincide with the earliest activity date (for the selected course) and the right edge of it must coincide with the latest activity date (for the selected course).

Both the X and Y axes of the plot must have labels. The padding on the left side of the plot region contains the Y labels: 0%, 10%, 20%, …, 100%. The padding on the bottom side of the plot region contains two X labels: the earliest date horizontally aligned with the left of the plot region and the latest date horizontally aligned with the right of the plot region.

[1] The number of pixels used for the padding may vary by one or two pixels due to rounding error.

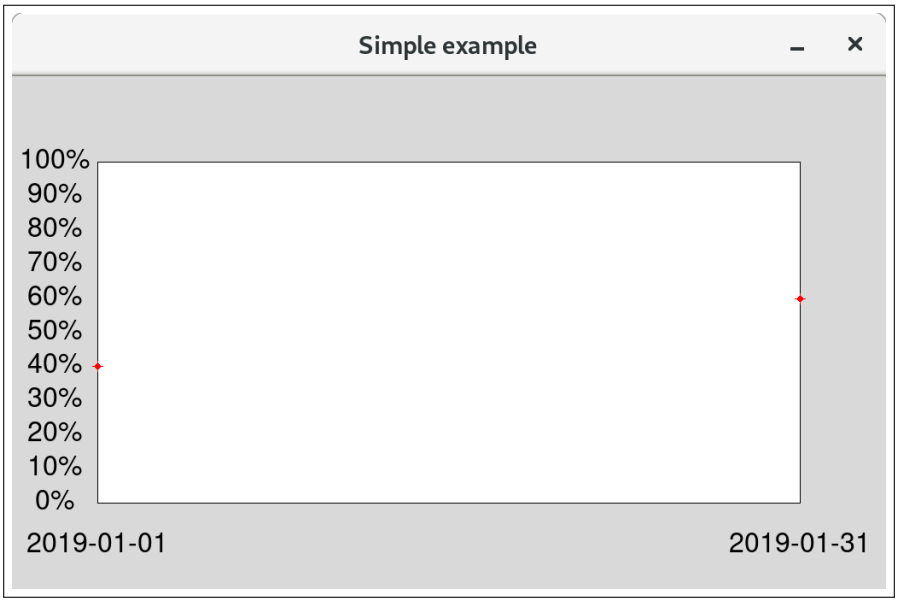
Within the plot, the image `point.gif` must be drawn at the position of each score for the course.

The user will close the window using the close functionality provided by the operating system, e.g., the X in the top corner of the window. Closing the window must not crash the program; instead, the program must return to the main menu.

Consider the following gradebook data:

```
Course: Simple example
  Activity: First activity (Weight: 50.0)
    Score: 40.0 (Completed: 2019-01-01)
  Activity: Last activity (Weight: 50.0)
    Score: 60.0 (Completed: 2019-01-31)
```

Plotting this course produces the following window:



Interaction

When the user clicks inside the plot region, text positioned within the padding at the top side must display the position of the click as

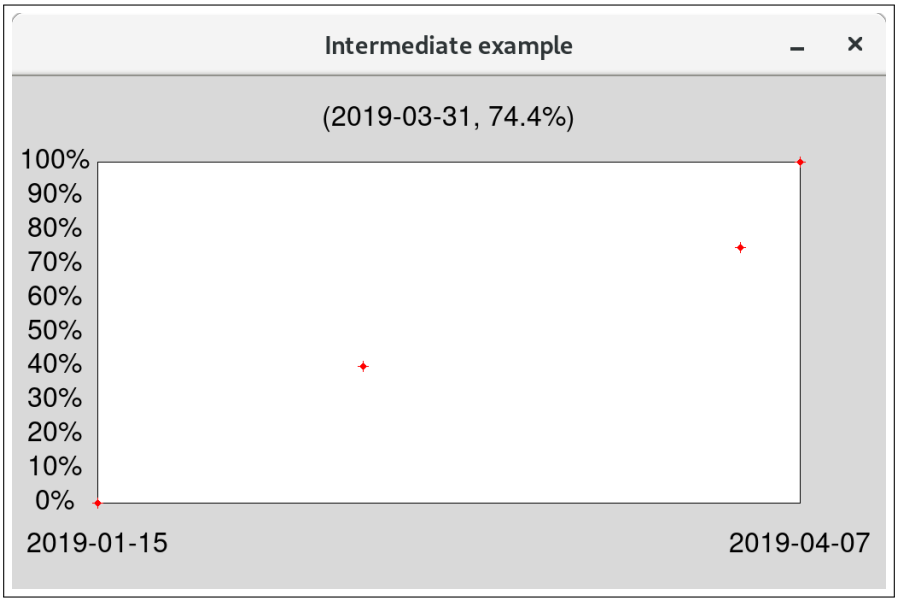
(*date*, *score*%)

where *score* is displayed to one decimal place (see the earlier hint: *Formatted string literals*). When the user clicks outside the plot region, the location text within the padding at the top side must disappear.

Consider the following gradebook data:

```
Course: Intermediate example
  Activity: First (Weight: 25.0)
    Score: 0.0 (Completed: 2019-01-15)
  Activity: Fourth (Weight: 25.0)
    Score: 100.0 (Completed: 2019-04-07)
  Activity: Second (Weight: 25.0)
    Score: 40.0 (Completed: 2019-02-15)
  Activity: Third (Weight: 25.0)
    Score: 75.0 (Completed: 2019-03-31)
```

Plotting this course and clicking near the score for March 31 (75.0%) produces the following window:



Deriving new dates

When you have a `datetime.date` object, it is possible to add a number of days to the object to derive a new `datetime.date` object. Unfortunately, you cannot simply add an integer. Instead, you need to create and add a `datetime.timedelta` object.

Consider the following example:

```
>>> date = datetime.date(2019, 1, 1)
>>> delta = datetime.timedelta(days = 5)
>>> new_date = date + delta
>>> print(type(new_date))
    <class 'datetime.date'>
>>> print(new_date)
    2019-01-06
>>>
```

Checklist

This checklist has been included to remind you of key requirements. Additional requirements may be described elsewhere in this specification.

- ☐ Updates menu option #7
  - ☐ Prompts for a course name
  - ☐ Returns to menu when given an empty string
  - ☐ Checks dates for course's graded activities
- ☐ Creates window
  - ☐ Creates with the size 1024×600 pixels
  - ☐ Includes title that reflects the course name
- ☐ Creates correct content inside window
  - ☐ Creates a rectangular plot region of 824×400 pixels inside the window
  - ☐ Pads the rectangle with 100 pixels on each of the four sides
  - ☐ Plots X labels for the X axis in the padding below the rectangle: the earliest date and the latest date
  - ☐ Plots Y labels for the Y axis in the padding to the left of the rectangle: 0%, 10%, 20%, …, 100%
  - ☐ **Plots scores in the correct locations**
  - ☐ Plots scores within the rectangle using the image `point.gif`
- ☐ Responds to mouse clicks within the rectangle
  - ☐ Displays the position in the padding above the plot rectangle
  - ☐ Displays the horizontal position (X) as a date
  - ☐ Displays the vertical position to one decimal place
- ☐ Responds to mouse clicks outside the rectangle
  - ☐ Removes (at least visually) the position displayed in the top padding
- ☐ Returns to the menu when the window is closed