10.014 CTD 1D: GPA Calculator

SC06 Group F

Andrew Yu, 1006879
Chen Chuxin, 1006964
Chen Yirong, 1004209
Tan Zorye, 1007199
Lucas Tan, 1006735

Documentation for GPA Calculator with Terminal Text UI

Files that make up the program:

1. main.py (Starting up the program, where the GPA Calculator program is called)
2. GPACalculator.py (GPA Calculator program)
3. mods.json (json file that contains information on all Term 1 modules (component names, scoring weightages, credit weightage)
4. Saved_Data (Folder of text files store user's data)

main.py

Under **if __name__ == '__main__':**

Only when main.py is explicitly and directly run (meaning not from another file), calls boot_up() and the program starts.

boot_up()

This function is activated after main.py is run, displaying 5 options for the user to choose from. These options are enumerated starting from 1 and the user can only input integers from 1-5. As our program only focuses on "GPA Calculator" for now, the other options are unavailable to the user.

start(choice: str)

This function starts the program the user has chosen. (Due to our program being only limited to "GPA Calculator" for now, only input 1 is available)

GPACalculator.py

introduce()

1. This function displays ASCII art of the GPA Calculator
2. Inside the try block, user is prompted to calculate or edit his GPA.
    a. ValueError Except block catches any non-numeric string inputs
        i. If user input is q, while loop terminates and user is directed back to the previous question
        ii. Else, displays "Invalid! Input only integers" to user, and user is directed back to 2.
    b. AssertionError Except block catches invalid numeric string inputs that are out of range of the available choices and displays "Invalid! Input 1 or 2" to user, and user is directed back to 2.
    c. FileNotFoundError Except block catches FileNotFoundError exception that would be raised in **edit_func** if the user wanted to find files to edit, but there were no existing files and displays "No files found!" to user, and user is directed back to 2.
3. If user input is 1, user is directed to **calculate_func**. Else (user input is 2), user is directed to **edit_func**

calculate_func()

1. Inside the try block, user is prompted to calculate his mod or term GPA
    a. ValueError Except block catches any non-numeric string inputs
        i. If user input is q, while loop terminates and user is directed back to the previous question
        ii. Else, displays "Invalid! Input only integers" to user, and user is directed back to 1.
    b. AssertionError Except block catches invalid numeric string inputs that are out of range of the available choices and displays “Invalid! Input 1 or 2” to user, and user is directed back to 1.
2. Inside the try block, user is prompted for his term he is in currently
    a. ValueError Except block catches any non-numeric string inputs
        i. If user input is q, while loop terminates and user is directed back to the previous question
        ii. Else, displays "Invalid! Input only integers" to user, and user is directed back to 2.
    b. AssertionError Except block catches invalid numeric string inputs that are out of range of the available choices and displays “Invalid! Input 1-10” to user, and user is directed back to 2.
3. If user input to 1. is 1, user is prompted for the name of the mod.
    a. If user input is q, while loop terminates and user is directed back to the previous question
    b. User is directed to **get_term_mods**
4. Else (user input to 1. is 2), user is directed to **edit_func**
    a. Inside the try block, user is prompted to calculate GPA from scratch (new) or use existing data
        i. ValueError Except block catches any non-numeric string inputs
            1. If user input is q, while loop terminates and user is directed back to the previous question
            2. Else, displays "Invalid! Input only integers" to user, and user is directed back to 4.
        ii. AssertionError Except block catches invalid numeric string inputs that are out of range of the available choices and displays “Invalid! Input 1-2” to user, and user is directed back to 4.
    b. If user input to 4. is 1, user is directed to **get_term_mods**. Else existing files that are visible and editable by the user (See **Saved_Data** section) will be used to calculate user’s term GPA. If the existing files are incomplete to calculate the full term’s GPA, the user will be directed to **get_term_mods** of the missing mods, before the user’s term GPA is displayed to them

edit_func()

This function uses fnmatch to access the saved text files to modify their content. Try and except pairings as above are also used extensively here to catch invalid inputs by the user.

1. Reads the content of the folder “Saved_Data” which contains the text files of stored score data of each module and also of the entire term.
2. Iterates through the folder and outputs to the user the files that are available for accessing, enumerated
3. Prompts the user to select a file to edit by entering an index
4. Opens the selected file and reads the content with mode=”r+”.
5. The text file is then fed through the function **text_to_dict** to format the string text into a dictionary form of data favourable for our program.
6. With text-processing and the function **display_components**, the data of the original saved string text is made sense of, enumerated and outputs on the console.
7. After accessing the file, the program prompts the user to choose a certain component to be edited with an index.
8. The program then prompts the user to key in the value for the component selected.
9. If the selected component is a valid index, run **calculate_component** on the component selected and update the score on that component.
10. The program then outputs the same data from point 6 again, with the updated value shown.
11. Overwrites the saved text file with the new updated data. This is done by first converting the updated dictionary data format back to a specified text string format using **dict_to_text**.

get_term_mods(term: str, mod: str = None)

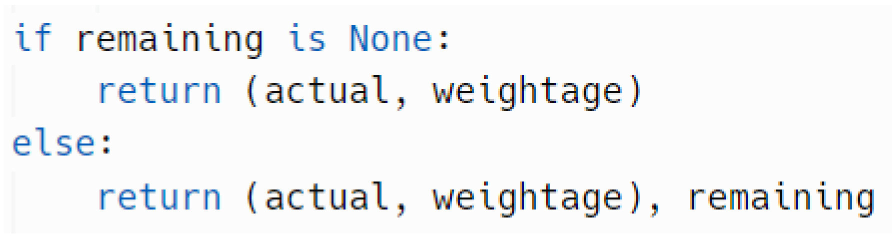
This function takes in two strings (term number and the module) and fetches the data pertaining to the mod from the mods.json file.

1. Opens and reads the mods.json file
2. Checks if the data of modules of the term exist
3. Checks whether the entered module exists in the data fetched from the file
    a. E.g. using text processing, we check if the input ‘module’ is either 10.013 or Math or any of Maths/Modelling/Analysis to determine which module data should be extracted from the .json file
4. If it exists, fetch the data related to the module and run **calculate_mod** on it.
5. If the module string was not specified, it implies that the program is not trying to calculate a specific module of a term. With how our program is structured, it means that the user wishes to calculate the GPA for his/her entire term and will therefore run **calculate_term** instead.
6. If the data does not exist, it will return the string “Unable to find mod”

calculate_component(component: str, weightage: int, remaining: int = None)

This function takes a component name (as a string, E.g. Midterms), the weightage of that component (as an integer) in the module, and has an optional parameter (as an integer) ‘remaining’ with default value = None. Within the function, the score is calculated and returned depending on what the input component is.

1. If the component is “Midterm” or “Finals”, the program will prompt for the actual score and max score to be entered, since these are the only components the user (student) will know the max score for, and the max score is likely not the weightage specified.
2. If the component is “Class Participation”, the program will call the function **give_survey(weightage)**. The score from the survey is then saved in a variable.
3. Else, the user will be prompted to input the actual/estimated score for a component of the module.
4. The function then returns a tuple containing a boolean and the score for the component.
5. If the input was ‘nil’, the variable ‘remaining’ will be assigned the integer value of the weightage. The function will return a tuple containing a tuple of the (boolean and the score for the component) and a variable remaining, which will be used in further calculations.



```
if remaining is None:
    return (actual, weightage)
else:
    return (actual, weightage), remaining
```

All inputs in this function go through data validation with our function **input_validated**. If any of the inputs are out of bounds or are not keyed in as ‘nil’, the program will prompt the user to try that again.

calculate_mod(term: str, term_mod: str, components: dict)

This function takes the term selected (E.g term 1), the module of the term to be calculated for and the components belonging to the module of choice.

This function prompts the user for his/her target score for the module, calculates the overall grade of the module by prompting the user for inputs for each component of the module, writes and saves a text file that acts as a database so that the program and the user still has access to them even after reopening the program (ensuring persistence of data), and outputs relevant information for the user in the terminal. If the user's overall score meets his/her target, the console will reflect as such and congratulate the user. If there was any component keyed as 'nil' because the user does not yet know his/her scores for that component (E.g before the Finals), the terminal will reflect what the user needs to score for all the remaining components which he has yet to know the scores for in order to meet the his target score

1. The function first prompts the user for a target score for the particular module.
2. The function iterates through all the components of a selected module with the components dictionary.
3. In this loop, the **calculate_component** function is used on every component of the module. The returned scores from each instance of **calculate_component** is then further processed and calculated.
4. At the end of the loop, an overall score in percentage for the module is calculated.
5. The value of the returned variable 'remaining' is accumulated for every 'nil' entered.
6. The console then outputs all the user's prompts and inputs thus far as a receipt to what was entered
7. If the user's entered scores for every component overall score met the target, the console will output as such and congratulate the user too. If the scores did not meet target, the console will output that it is impossible for the user now and to try harder next time.
8. If the user entered 'nil' for any component, the formula will be applied to find out what the user needs to score for all those remaining components in terms of a percentage, in order to reach the user's target score.
   Percentage needed for remaining components = (target - overall score attained)/(max remaining score to be scored) * 100%
9. The function then calculates the gpa for the module (out of 5.0) using the function **calculate_gpa**, then prints it out on the console. The function **calculate_gpa** also returns the credit score contributed to the overall term GPA by the module with the given scores.
10. Lastly, the function creates a file containing all the prompts and respective inputs entered by the user using the function **save_data**. The file name contains the name of the module for identification.

calculate_term(term: str, term_mods: dict)

This function takes a term as a string (E.g. ‘1’) and a dictionary containing all the modules and related components under the respective term.

1. Iterate through each module in the term.
2. **calculate_mod** is then used on each module of the term, and the total credit scores from each module is summed up in an accumulator variable.
3. A string stores and concatenates the results of each module.
4. At the end of the loop, create a file using the function **save_data** which contains all the entered data for the term.
5. Outputs to the user the calculated GPA for the term.

calculate_gpa(max_credits: int, current_pct: float = None, current_credits: float = None) -> tuple

This function takes max_credits as an integer, an optional parameter ‘current_pct’ (as float) with default value = None, and another optional parameter ‘current_credits’ (as float) with default value = None, and returns a tuple of current_gpa, current_credits and max_credits. For a successful calculation, max_credits and either of the optional parameters must be supplied, as gpa calculation relies on 2 of those arguments.

As we have no access to the exact percentage - grade - gpa system, we can only approximate/estimate by

current_credits = (current_pct / 100) * max_credits, then rounded to 2.dp if current_credits is not supplied but current_pct is.

Then current gpa = (current_credits / max_credits) * 5.0, then rounded to 2.dp

input_validated(qn: str, bounds: tuple)

This function takes a question as a string that will be asked to the user and bounds as a tuple to check if the user’s input is valid and within the specified bounds.

Using try-except blocks, input is verified that it can be converted to type float and it is within the specified bounds. If the input is ‘nil’, the ValueError except block catches it and returns a boolean False that would be used for further computations.

give_survey(total_percentage: int) -> float

This function takes a total_percentage (as an integer) that is the weightage of class participation in that component and returns a float value that is the class participation percentage attained.

This function prompts the user with a predetermined set of questions that will help gauge his class participation grades in terms of percentage. This set of 4 questions requires the user to input an integer on a scale ranging 0-10.

advise(grades: dict) -> str

This function takes in “grades” (of type dictionary), and returns a dynamically generated advice listed from easy to hard (of type string), based on the module and user’s grades. If a user fails to attain his desired score (target), he will be advised to make up the difference with the remaining components that he can still work on in future to achieve his target at term’s end.

dict_to_text(grades_dict: dict) -> str

This function takes in “grades_dict” (of type dictionary), and returns the dictionary converted into string as specified to save the data appropriately in .txt files.

text_to_dict(grades_text: str) -> dict

This function takes in “grades_text” (of type string) and returns the string converted into a dictionary as specified to use in future computation upon reading the .txt files.

display_components(term_mod:str, components:dict, enumerated:bool)

This function displays nicely formatted components for any given module.

save_data(file_name: str, grades_text:str)

This function takes in the user’s most recent input for each module and stores it as a .txt file under the specified file_name.This saved text data allows the user to update individual components of previously entered data in order to calculate his/her overall term GPA without the need to re-enter them even if the program restarts (ensuring persistence of data).

E.g. Before attempting finals, the entry for Finals will be nil. After knowing the score for finals, the user can revisit the program and need only update his/her actual score for finals before calculating the term GPA

mods.json

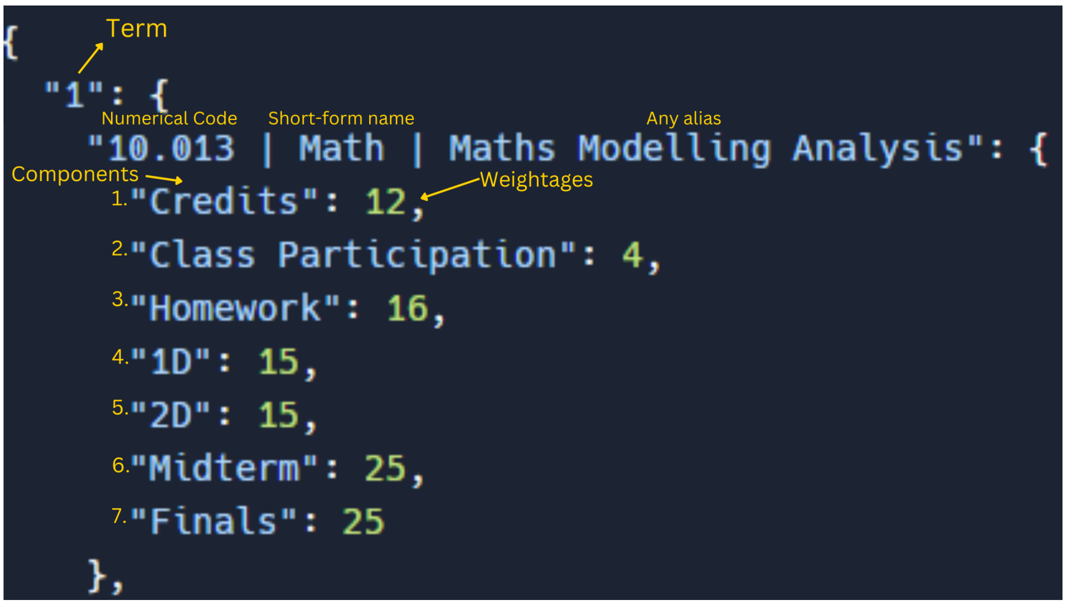
Data of mods is stored in the following format

Term, Components are strings

Numerical code, Short-form name, Any alias is a string of which each is separated by |

Weightages are integers

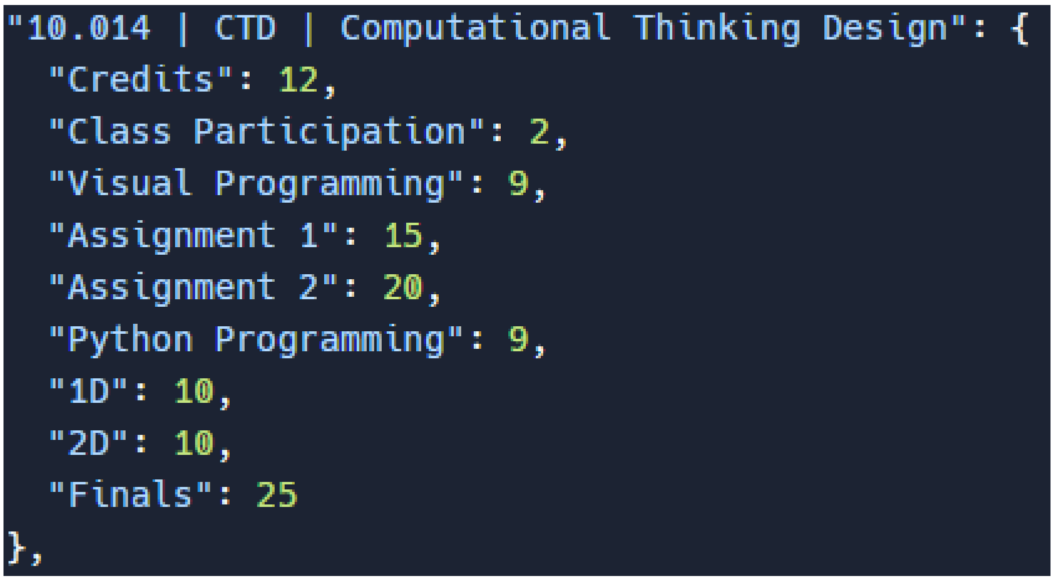
Components are to be in this exact order



```
{
  "1": {
    "10.013 | Math | Maths Modelling Analysis": {
      "Credits": 12,
      "Class Participation": 4,
      "Homework": 16,
      "1D": 15,
      "2D": 15,
      "Midterm": 25,
      "Finals": 25
    },
```

If some modules do not have these specific components, they must still follow this order wherever possible

Eg.



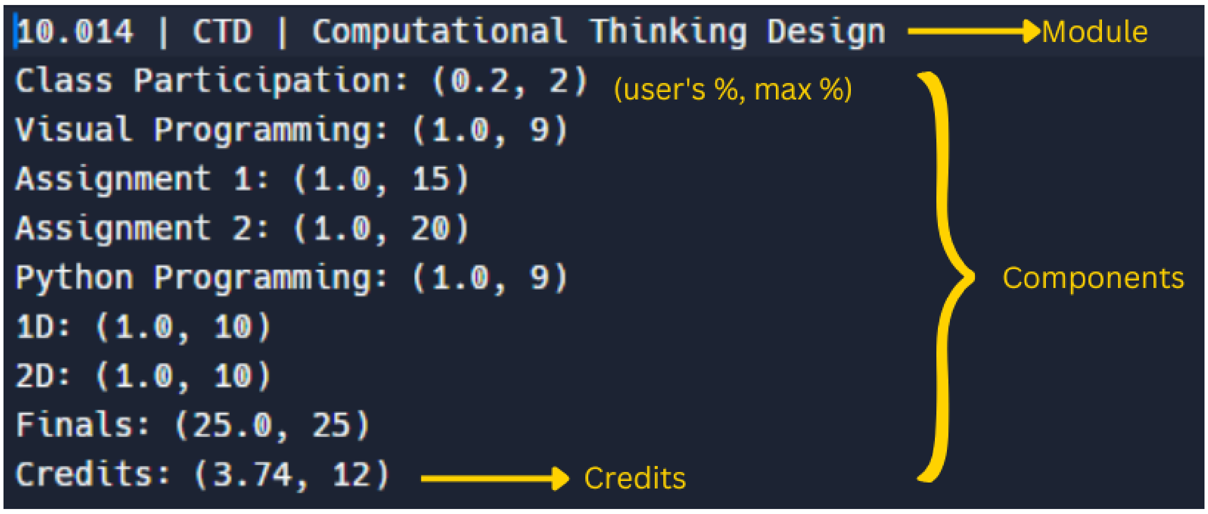
```
"10.014 | CTD | Computational Thinking Design": {
  "Credits": 12,
  "Class Participation": 2,
  "Visual Programming": 9,
  "Assignment 1": 15,
  "Assignment 2": 20,
  "Python Programming": 9,
  "1D": 10,
  "2D": 10,
  "Finals": 25
},
```

Saved_Data

A folder, initially empty as it stores no user data.

Stores module data as Eg. “term_1_ctd.txt”, where its contents can be edited and viewed by the user

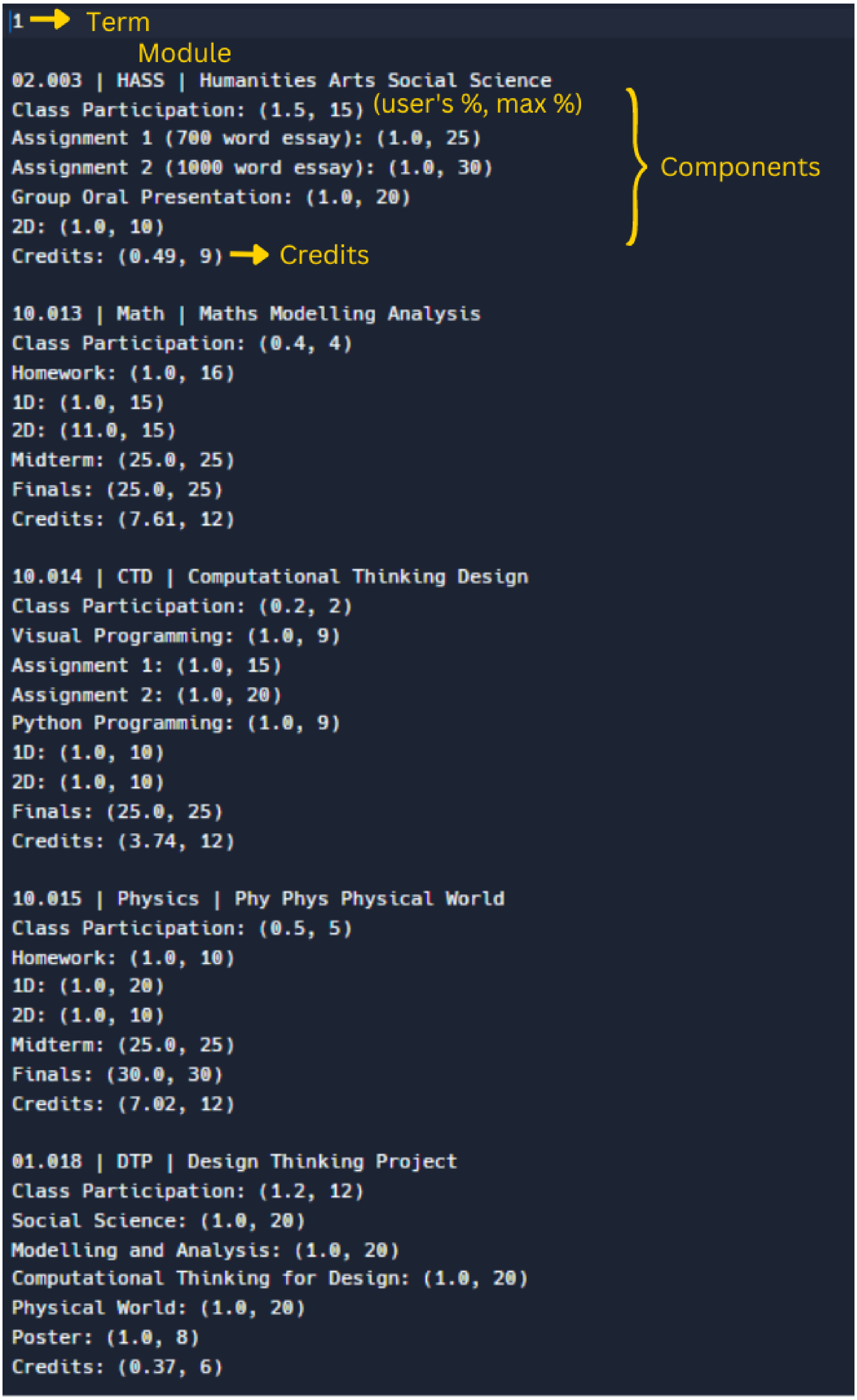
Eg. of contents inside follows the format as such



```
10.014 | CTD | Computational Thinking Design
Class Participation: (0.2, 2)
Visual Programming: (1.0, 9)
Assignment 1: (1.0, 15)
Assignment 2: (1.0, 20)
Python Programming: (1.0, 9)
1D: (1.0, 10)
2D: (1.0, 10)
Finals: (25.0, 25)
Credits: (3.74, 12)
```

Stores term data as Eg. “_term_1.txt”, the underscore indicating its contents should not be edited or viewed by the user

Eg. of contents inside follows the format as such, which is the combination of all content from the module data files



```
1
02.003 | HASS | Humanities Arts Social Science
Class Participation: (1.5, 15)
Assignment 1 (700 word essay): (1.0, 25)
Assignment 2 (1000 word essay): (1.0, 30)
Group Oral Presentation: (1.0, 20)
2D: (1.0, 10)
Credits: (0.49, 9)

10.013 | Math | Maths Modelling Analysis
Class Participation: (0.4, 4)
Homework: (1.0, 16)
1D: (1.0, 15)
2D: (11.0, 15)
Midterm: (25.0, 25)
Finals: (25.0, 25)
Credits: (7.61, 12)

10.014 | CTD | Computational Thinking Design
Class Participation: (0.2, 2)
Visual Programming: (1.0, 9)
Assignment 1: (1.0, 15)
Assignment 2: (1.0, 20)
Python Programming: (1.0, 9)
1D: (1.0, 10)
2D: (1.0, 10)
Finals: (25.0, 25)
Credits: (3.74, 12)

10.015 | Physics | Phy Phys Physical World
Class Participation: (0.5, 5)
Homework: (1.0, 10)
1D: (1.0, 20)
2D: (1.0, 10)
Midterm: (25.0, 25)
Finals: (30.0, 30)
Credits: (7.02, 12)

01.018 | DTP | Design Thinking Project
Class Participation: (1.2, 12)
Social Science: (1.0, 20)
Modelling and Analysis: (1.0, 20)
Computational Thinking for Design: (1.0, 20)
Physical World: (1.0, 20)
Poster: (1.0, 8)
Credits: (0.37, 6)
```

Documentation for Term 1 GPA Calculator with GUI powered by tkinter

This program uses the tkinter library to create a GUI for the Term 1 GPA calculator

create_layout(components: list, frame, subj: str).

This function creates the layout of labels and entry boxes belonging to each module. It takes in a list of component names with respect to a particular module, a parent frame widget to create the layout in and the name of the module(subj).

Within the function, it dynamically creates the labels that displays the text of each component in the module, the entry boxes for users to key in their scores, the label that serves as an output display screen where we display our text and results to the user and a button meant to initialise the calculation function. These widgets are organised using the grid method of tkinter. The function then returns a dictionary with specific keys and the values stored are the widgets themselves.

calculate_component(module: dict, weightages: dict).

This function takes in a dictionary of widgets to access and a dictionary that contains the weightages of each scoring component in a particular module.

The pseudocode is as follows:

For every component in the module:

1. Retrieve the scores entered in the entry boxes of the related module
2. If the values are neither 0-100 or ‘dk’, display a message to the user to highlight that something is wrong and stop the calculation function
3. Calculate the total score / required score to achieve target based on inputs
4. Output a message to the user to show results
5. Return the total score calculated, saved in a dictionary (credit_scored) as a value, tagged to the module’s name as the key for calculation of the total CGPA when needed

calculate_CGPA().

This function takes all the scores saved in the dictionary credit_scored and calculates the CGPA based on the credit weightages of each module.

```
class Module:

    def __init__(self,mod_entry,mod_scores,name):
```

```
  def calculate(self):
```

This class creates an object that contains a dictionary of widgets, the scoring weightages and name of the module. It also contains the function **calculate(self),** which calls for the **calculate_CGPA** function, inputting the dictionaries pertaining to the module for calculations.

Under **if __name__ == '__main__':**

Only when main.py is explicitly and directly run (meaning not from another file), all the frame widgets and widgets of the CGPA section are created. It calls for the **create_layout** function to create the layouts for each module and creates an object of class Module for each different module. (Math, Physics, CDT, HASS, DTP 1) It then packs all the widgets using a combination of the pack and grid methods of tkinter to create the layout shown.

Window.mainloop().

Opens the GUI