**AUTOMATIC PDF GENERATOR**

(A multi-document compiler)

**Minor Project - II**

Efforts by:

**Prachi Chauhan**

**Abhijeet Sanjiv Bonde**

**Abhishek Pundir**

Under the Esteemed Guidance of

**Mr. Ankit Vidyarthi**



**Department of CSE/IT**

**Jaypee Institute of Information Technology University, Noida**

**May 2022**

**Table of Contents**

*List of Tables*

*List of Figures*

*Acknowledgement*

*Declaration*

*Certificate*

**Introduction**

**Problem Statement**

**Requirement Analysis**

**Proposed Workflow**

**Implementation**

**Conclusion of the Report and Future Scope**

**References**

**List of Figures**

**Figure title**

2.1 Flowchart of the project

2.2 Flowchart of internal processing of values and variables (data handling)

2.3 Flowchart of file generation throught the process

3.1 Execution of main.py

3.2 Execution of COs.py

3.3 Execution of main2.py

3.4.1 Execution of student\_details.py (main menu)

3.4.2 Execution of student\_details.py (CO’s input window)

3.5 Execution of WriteWord.py

3.6 Execution of Attainment.py

3.1 Execution of MergeWord.py

**ACKNOWLEDGEMENT**

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. We take this opportunity to express our profound sense of gratitude and appreciation to all those who helped us throughout the duration of this project. We would like to express our special thanks of gratitude to our mentor Mr Ankit Vidyarthi, who gave us the golden opportunity to be a part of this interesting project and for providing guidance and expert supervision for this project. We are really thankful to him. Our thanks and appreciations also go to our college mates in developing the project and people who have willingly helped us out with their abilities. We are making this project not only for fulfilling a college requirement but to also enhance our knowledge. We express our sincere gratitude towards each and every one who helped us towards the completion of the Project.

**DECLARATION**

We Prachi Chauhan, Abhijeet Sanjiv Bonde, and Abhishek Pundir hereby declare the following usage of the open-source code and prebuilt libraries in our minor project in 6th Semester with the consent of our supervisor. We also measure the similarity percentage of pre written source code and our source code and the same is mentioned below. This measurement is true with best of our knowledge and abilities.

1. List of pre-build libraries
2. List of pre-build features in libraries or in source code.
3. Percentage of pre-written source code and source written by us.

|  |  |  |
| --- | --- | --- |
| Student ID | Student Name | Student signature |
| 19103002 | Abhijeet Bonde |  |
| 19103309 | Prachi Chauhan |  |
| 19103016 | Abhishek Pundir |  |

**CERTIFICATE**

This is to certify that the work titled “Automatic pdf Generator” submitted by Abhijeet Bonde, Prachi Chauhan and Abhishek Pundir of B. Tech of Jaypee Institute of Information Technology, Noida has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of any other degree.

Digital Signature of Supervisor

Name of Supervisor: Mr. Ankit Vidyarthi

Designation Date

**Introduction**

Automatic pdf generator is a python-based software which is used to compile both static and dynamic documents into one single pdf automatically. It mainly focuses on creating a single course file for JIIT faculty. Since it's so inefficient for the same data to be stored in different files, causing redundancy and hence wastage of space, a more effective solution is required.

The objective of this project is to create such a software. It can take multiple documents as an input, and generate a single PDF file that contains the compilation of all the data inside the inputted files.

So if a teacher would need to fetch some information, they will have a compilation of the complete data in a single PDF, therefore saving time and effort.

Apart from taking input, this software will also have a feature of storing static and dynamic documents in a sequential manner.

**Problem Statement**

The abundance of different files and types has indeed diversified the data and it's storage. However, when it comes to traversal and access, it is still somewhat tedious to find data that is stored in parts in many different files. For example, student data, grades, lesson plan, attendance etc. are all stored separately. Hence, if a teacher needs to assess a student, they'll have to open multiple files and compare data by keys in order to extract the needed information.

Our software is the key solution to the above problem statement. By creating a single Course pdf having 20 sub documents from Mission Vision JIIT to course closing report.

**1. Proposed approach/Workflow**

**1.1** **Importing the Module**

To begin working on the application, we will first have to import the necessary module.

It would allow us to use the constants and classes by their names instead of qualifying

them. We’ve assigned our window the variable "root”.

**1.2 Adding Buttons**

Now that we have our Tkinter running, we can add buttons to our program.

Buttons are the simplest section of any GUI program. The window we created in the last section is the parent for the buttons we made now. That’s because we need to add a grid or canvas and a pack to our program. They are referred as geometry managers.

**1.2.1 Adding Entry, Label, Radio buttons and more…**

More such widgets similar to Button are also added for different functionality, like Entry widgets are used to take a string input from the user, Label widgets are used to put a heading or label on top of a widget specifying the purpose of the lower widget, Radio Buttons are added to take only one of more options as an input from the user.

**1.3 Implementing the Pack Manager**

We’ll use the pack and grid method to add our buttons and labels to the parent widget. With the pack manager, we can tell which side of the parent widget you want our child widget to be packed against.

**1.4 Making the Buttons Useful**

To make our buttons perform specific functions, we’ll have to use the ‘configure’ function. We can pass any keyword argument to a widget’s ‘configure’ method to pass it while creating it. The ‘command’ parameter ties the buttons to callback functions when we create them or add ‘configure’ to them.

**1.5 Temporary Storage variables**

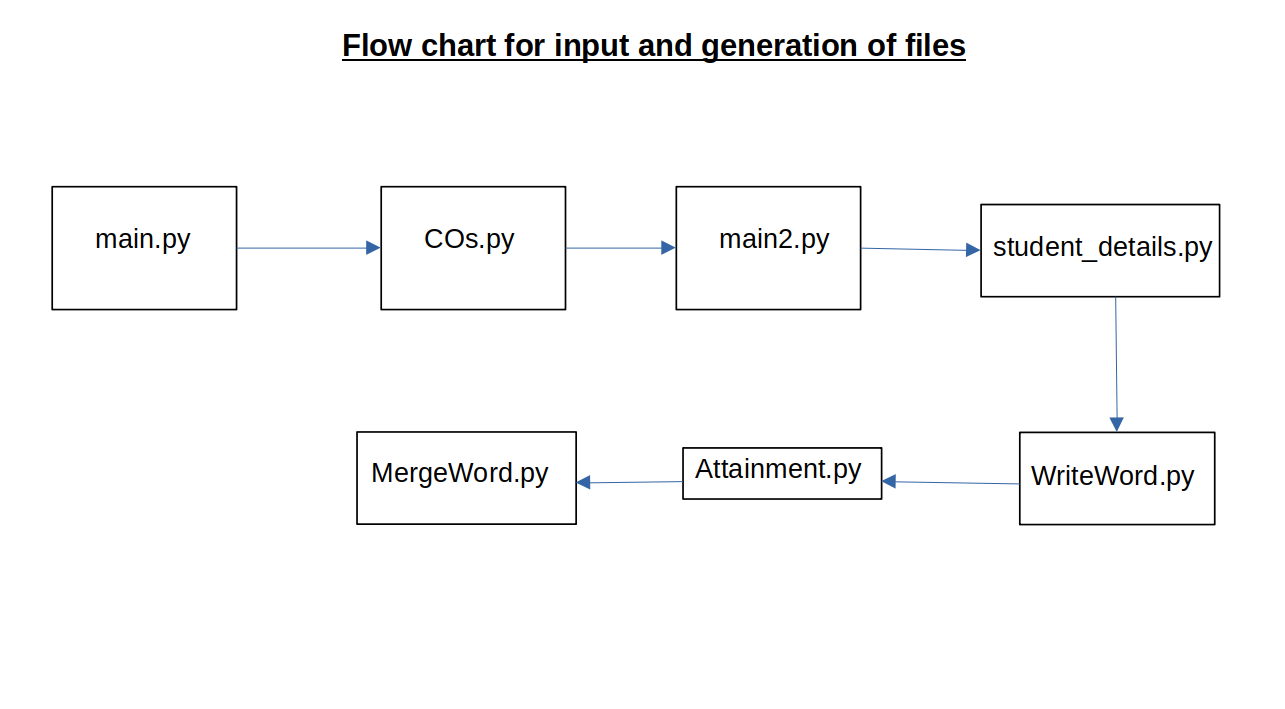
Once all the buttons are created, different type of widgets need different type of variables to store the values. For this purpose we use temporary variables that will hold the user selected value during the execution of the module.

**1.6 Permanent Storage (json)**

After the execution of the module, values have to be stored in a specific format due to redundant usage of these values. For this purpose we have used **json** files.

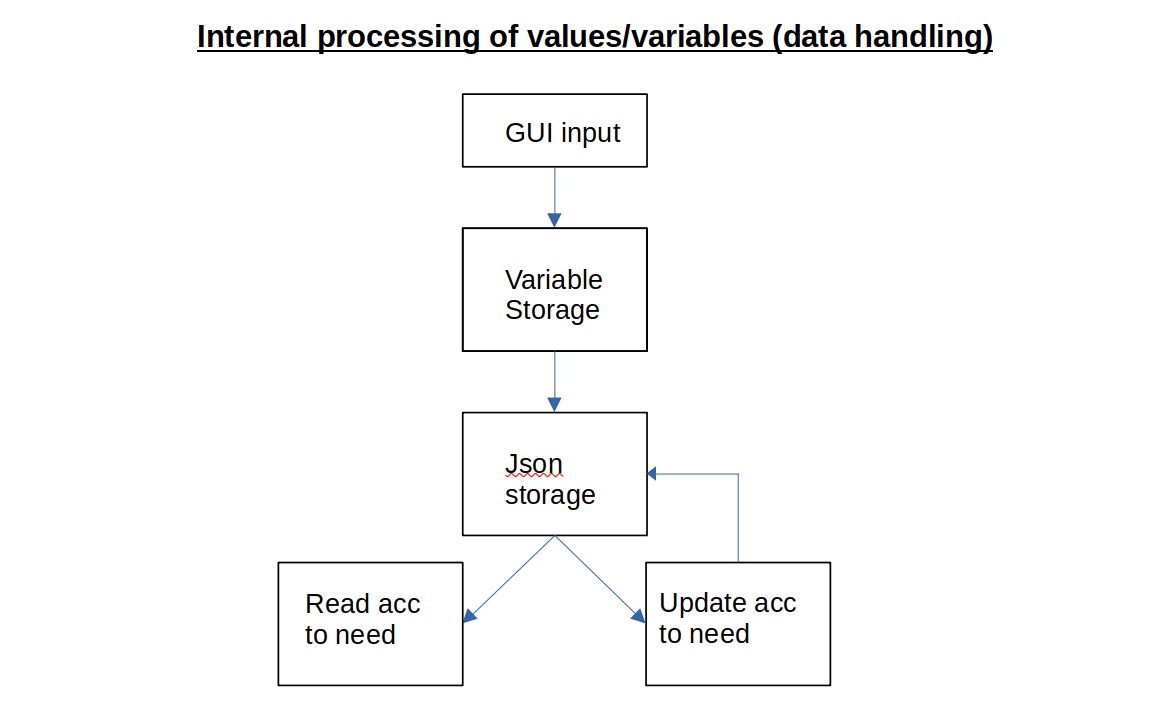
**2. WorkFlow (moduleWise)**

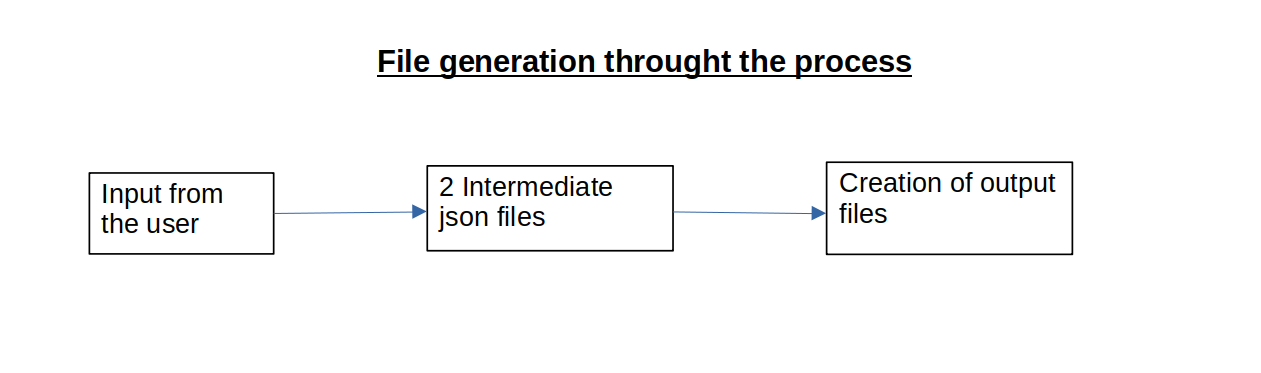
**Attached below is the flow chart of the working of program (Filewise).**

****

**The following flowchart displays the internal trasnfer of values as**

**for the ease of processing**

****

****

**2. Libraries used**

1. **json**

This library is used to read and write json intermediate files.

1. **tkinter (ttk widgets)**

This library is used to make the main GUI and its subsidary parts. The main category of widgets used from this library is **ttk**.

1. **Docx**

This library is used to write data to docx file using different stylings and attributes.

1. **Tkcalendar**

This library is used for displaying the calender in tkinter format in a GUI format.

1. **Xlwt**

This library is used to write data to xlsx spreadsheets using different stylings and underlining features.

1. **Pandas**

This library is used to read the data from xlsx file for furthur processing.

1. **Docxcompose**

This library is used to merge 2 word documents.

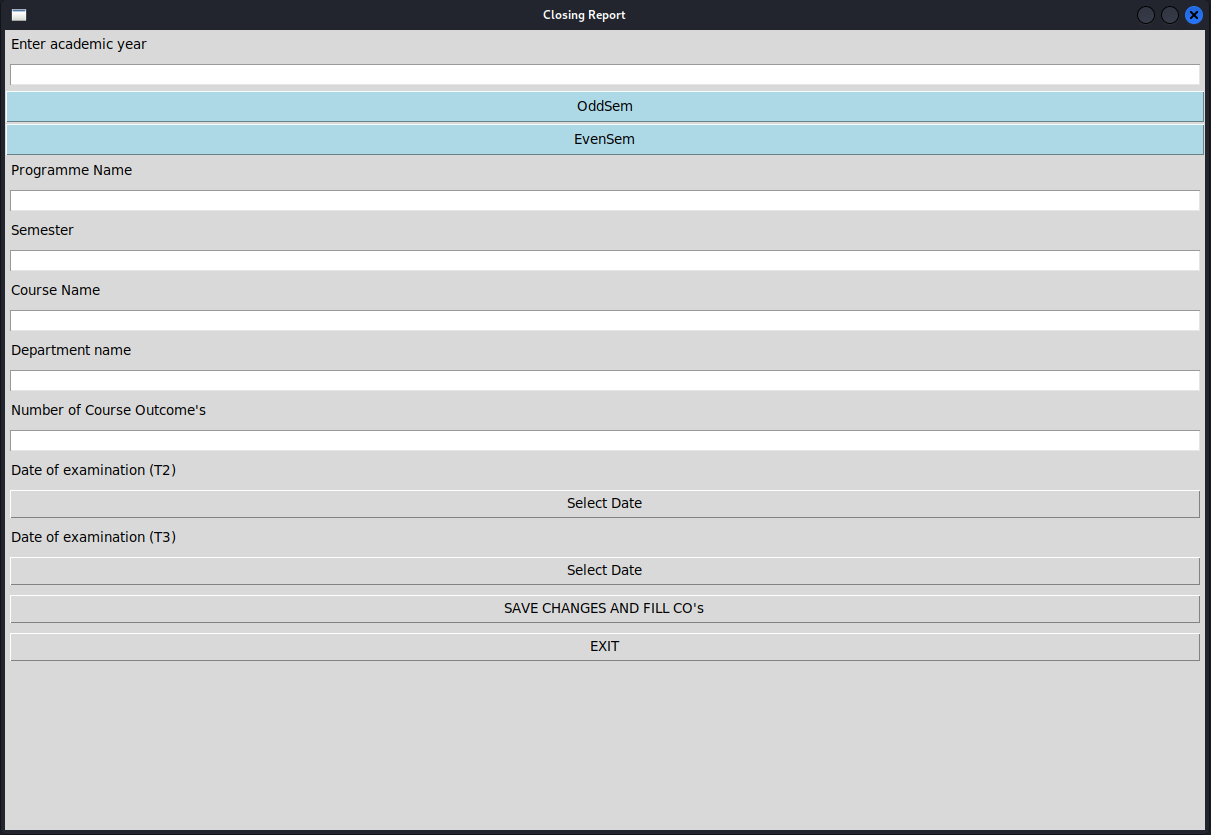
**3. Implementation**

**3.1 main.py**

This module takes the following **9** inputs

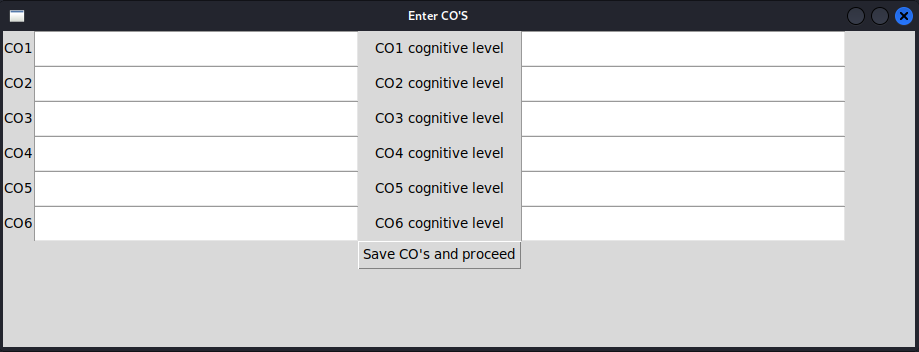
1. Academic year (String)
2. Odd/Even semester (Radio button)
3. Programe name (String)
4. Semester (String)
5. Course name (String)
6. Department name (String)
7. Number of course outcomes (Integer)
8. Date of T2 examination (Datepicker)
9. Date of T3 examination (Datepicker)

and stores these inputs in variables and then creates a json file named as **“details.json”** and stores the values from the variables in standard dictionary format which is created for internal processing. A Screenshot of the module is as attached below.

****

**3.2 COs.py**

After clicking the **“SAVE CHANGES AND FILL CO’s”** button the next file from the flowchart comes into action (is Cos.py). This module input from the user as **Course Outcomes** and their **Cognitive Levels.** The number of Entry widgets depend on the value user has entered in the previous module in the field of Number of Course Outcomes, and after clicking the **“Save CO’s and proceed”** button it appends the values of the Entry widget to the same json file created by the module before (main.py) according the same internal dictionary format as in previous module.

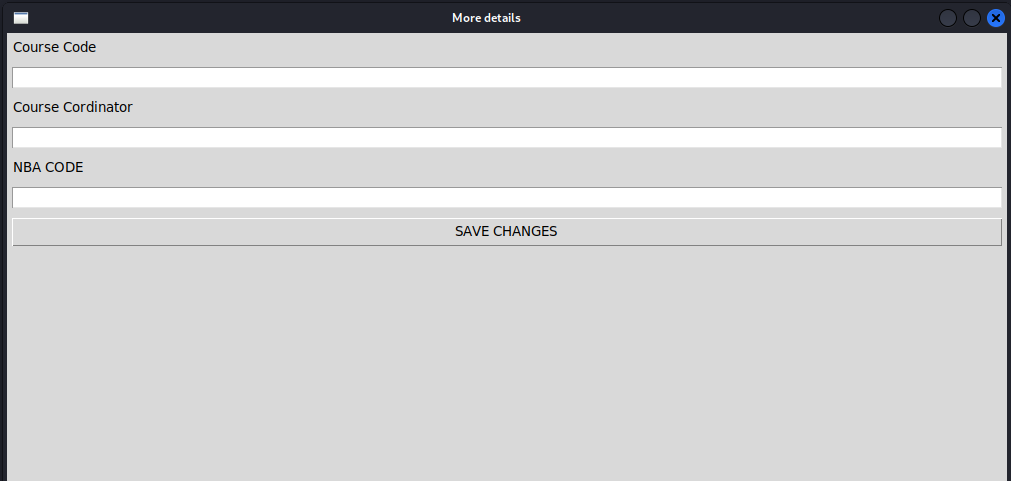
Attached below is a screenshot of the Cos.py module

**3.3 main2.py**

In this module there are 3 inputs taken from the user

1. Course Code (String)
2. Course Cordinator (String)
3. NBA Code (String)

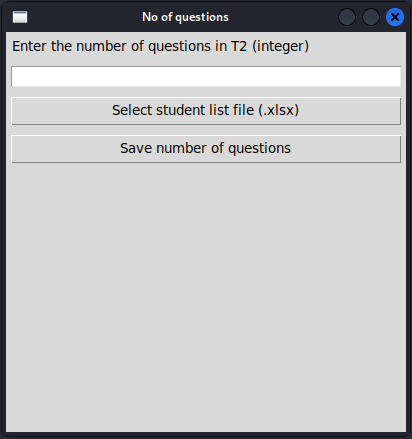
and after clicking the **“SAVE CHANGES”** button it appends the values of the Entry widget to the same json file created by the module before (main.py) according the same internal dictionary format as in previous module.

****

**3.4 student\_details.py**

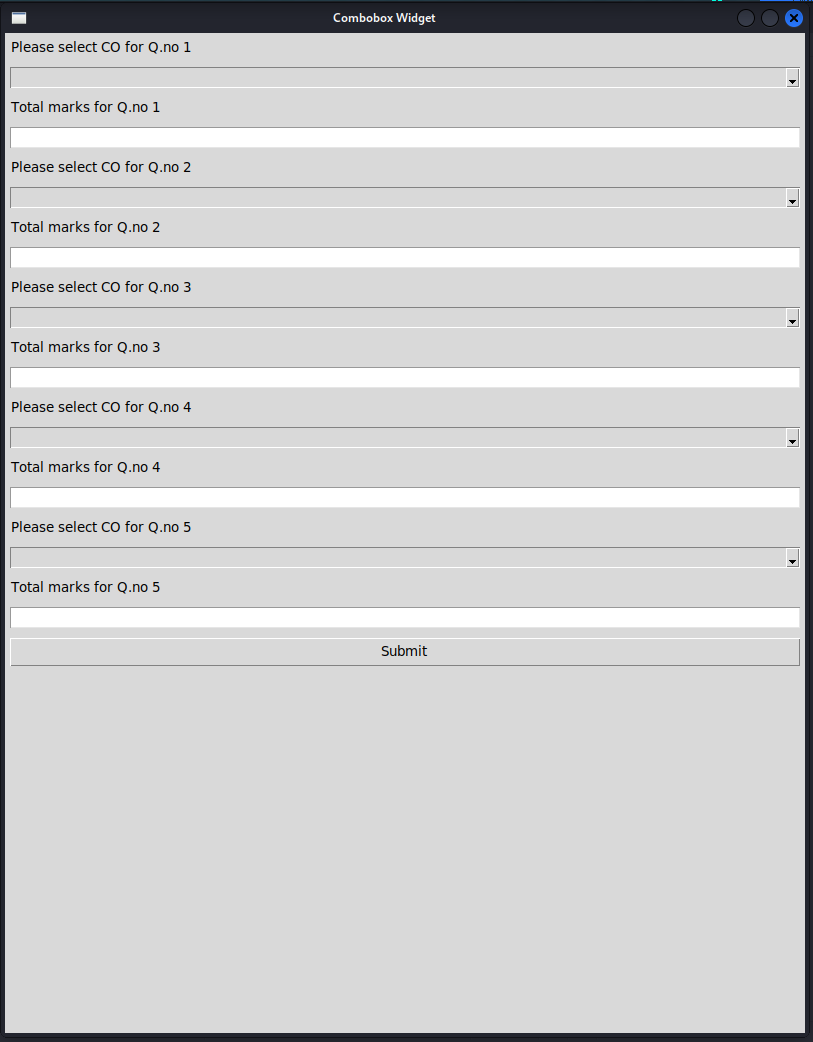
This module takes 2 inputs

1. Number of questions in a particular examination (Integer)
2. List of students (.xlsx file)

****

Here the user has to give the list of students with their enrollment number to the app in 2nd requirement in xlsx format.

after clicking the **“Save number of questions”** button it appends the values of the Entry widget to the same json file created by the module before (main.py) according the same internal dictionary format as in previous module and also creates a new json file named as **“student\_details.json”** which has different dictionaries for names and enrollment numbers. Now, depending on the value of **“Number of questions in T2”** the next window containing Course outcome and marks for that question appears for the user to fill in details.

****

After the user clicks **“Submit”** button it appends the values of the Entry widget to the same json file created by the module before (main.py) according the same internal dictionary format as in previous module.

**3.5 WriteWord.py**

This module reads the required details from **“details.json”** and creates a new document (.docx) and writes the the read data and also writes more headings and titles also it creates table.



This module does not have any output on the terminal window as it only reads and writes the data.

**3.6 Attainment.py**

This module initially reads the data from **“details.json”** and then writes it to a .xlsx file and also with it writes some headings and titles. These are just the static and para-dynamic data that has been written till now.

In the next step it reads the student list from **“student\_details.json”** file and according to the length of one dictionary it makes columns in table format and writes the student names and their respective enrollment numbers.

In one of the code segments it calculates the total marks according to CO’s (input from **section 3.4**) that are present in a particular examination and writes them to the heading of table previously created which is a dynamic data. The basis of summation is based on the Course Outcomes. Also it writes some the criteria for the judgement of attainment in the top right corner which is static data.

****

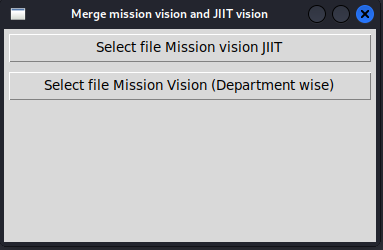
This module prints the total marks according to CO’s which are written to the .docx file

**3.7 MergeWord.py**

The sole purpose of this module is to combine 2 static word documents which are

1. **Mission vision JIIT**
2. **Mission vision (Department wise)**

This module takes 2 mentioned file as input and gives a other merged word file as output.



The library used for combining these 2 word documents is **“docxcompose”.**

**Conclusion of the report and future Scope**

We have embarked on developing a pdf Compiler for both static and dynamic documents. This project describes the making and developing the software. Currently our initial compiler can take input as static files, extract data from user for dynamic files and include in the final pdf. We can extend the compiler in both its ability to handle more a wide range of programs and add more features like taking input as an image or voice format. Adding desirable theme to the final pdf. Adding features like conversion of images to pdf. More advance features for dynamic documents. Generation of reduces size pdf for occupying less space.

**References**

**[ 1 ]** <https://anvil.works/blog/generate-pdf-with-python> *by Meredydd*

**[ 2 ]** Python GUI Programming with Tkinter: Design and build functional and user-

friendly GUI applications, 2nd Edition 2nd ed. Edition *by Alan D. Moore*

**[ 3 ]**

**[ 4 ]**