Relational Division Operator

By: -
Karandeep Singh Sehgal
Btech CSE - Section A
A50105220029

What is relational division operator ?

A(x, y) / B(y) = it results x values for that there should be a tuple <x, y> for every y value of relation B.

Relation A(x, y)

A	c1
B	c1
A	c2
C	c2

÷

Relation B(y)

c1
c2

=

A

A(x, y)/B(y)

Implementing Relational Division Operator in PYTHON

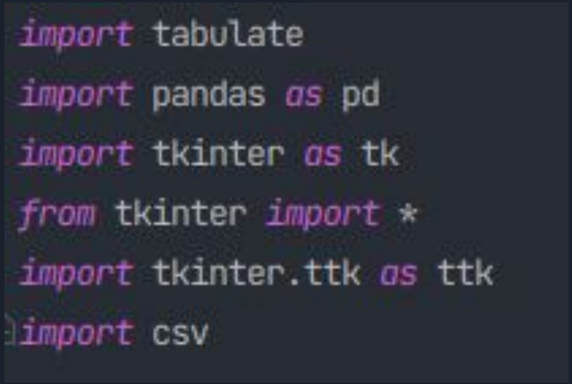
Packages Used

- Pandas and csv packages are used to manage csv files and dataframes.

- Tabulate is used to display dictionaries and list in a tabular and nicer way.

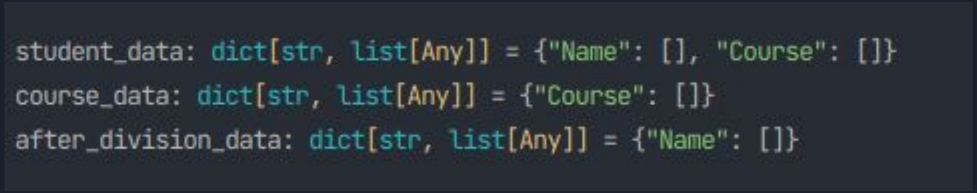
- Tkinter is used to create GUI windows.

  Tkinter.ttk is an extension package of tkinter which include a lot more widgets (like TreeView).



```
import tabulate
import pandas as pd
import tkinter as tk
from tkinter import *
import tkinter.ttk as ttk
import csv
```

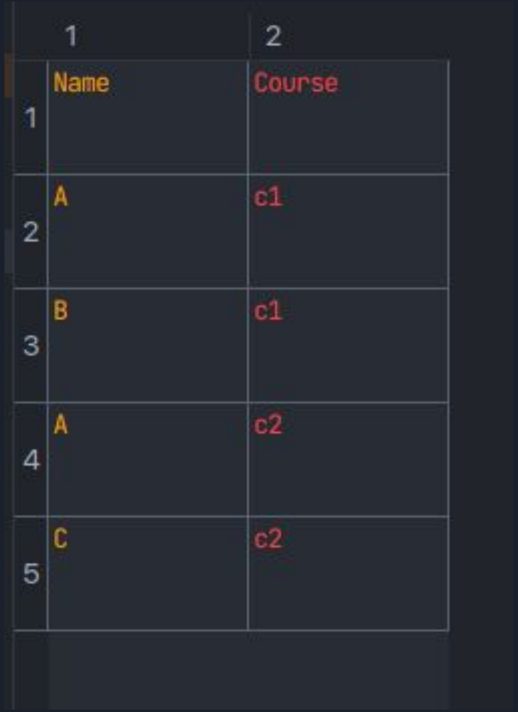
Dictionaries Created


```
student_data: dict[str, list[Any]] = {"Name": [], "Course": []}
course_data: dict[str, list[Any]] = {"Course": []}
after_division_data: dict[str, list[Any]] = {"Name": []}
```

- student_data : It contains the data of each student with his/her respected course.
- course_data: It contains all the courses present in the university.
- after_division_data: It contains the name of the students who have opted for all of the courses present in the course_data.

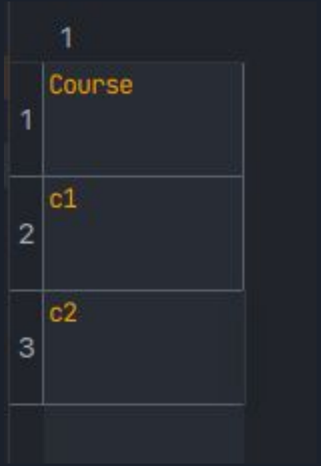
  OR

  The result of relational division operation performed on student_data and course_data student_data.csv :



Name	Course
A	c1
B	c1
A	c2
C	c2

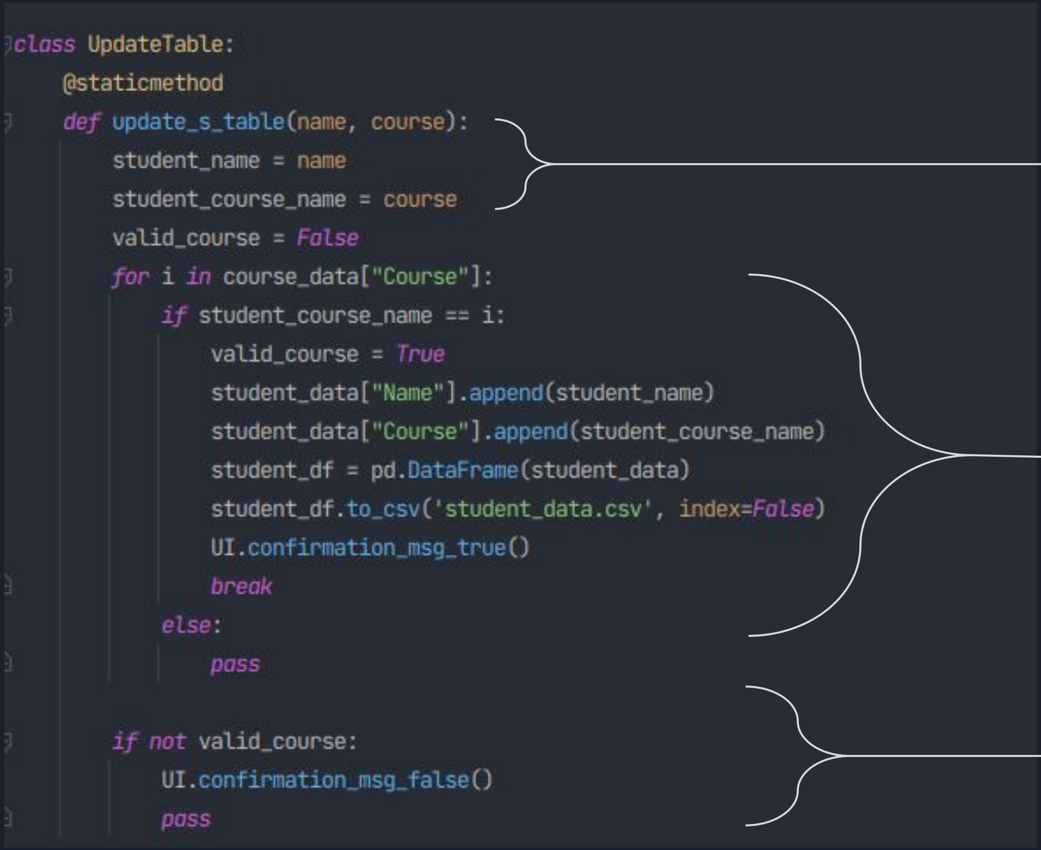
Course_data.csv :



Course
c1
c2

UpdateTable class

Method for updating the student_data relation



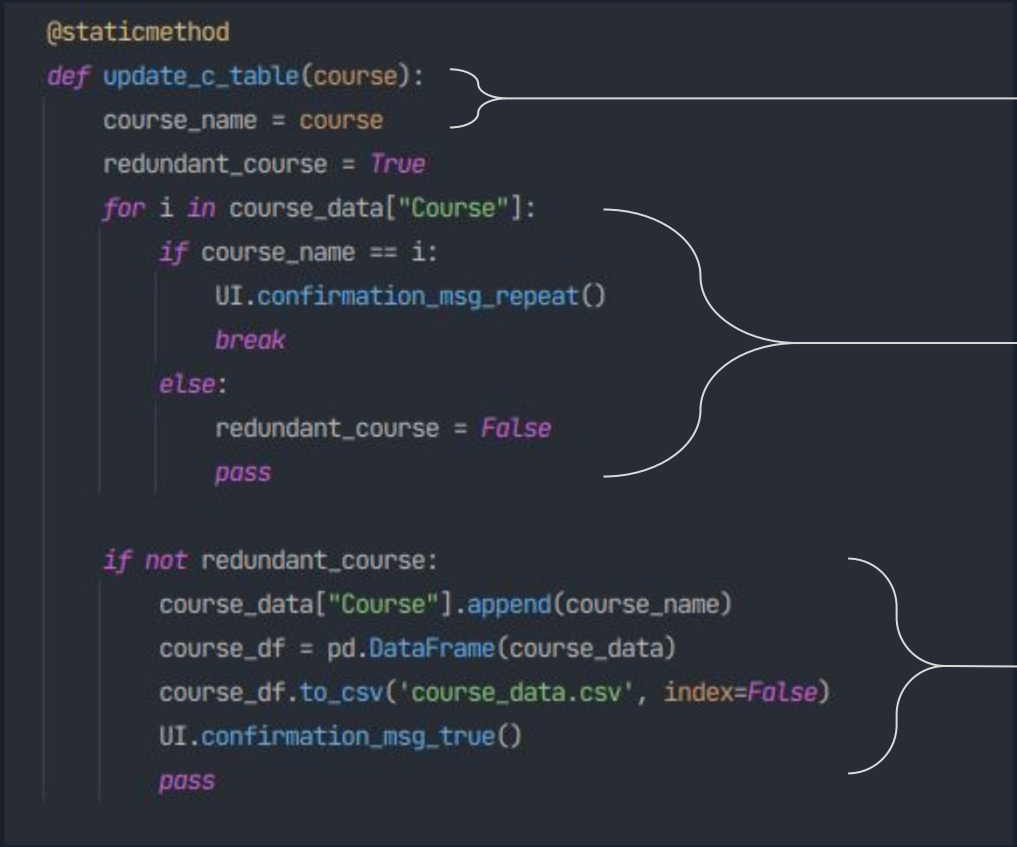
```
class UpdateTable:
    @staticmethod
    def update_s_table(name, course):
        student_name = name
        student_course_name = course
        valid_course = False
        for i in course_data["Course"]:
            if student_course_name == i:
                valid_course = True
                student_data["Name"].append(student_name)
                student_data["Course"].append(student_course_name)
                student_df = pd.DataFrame(student_data)
                student_df.to_csv('student_data.csv', index=False)
                UI.confirmation_msg_true()
                break
            else:
                pass

        if not valid_course:
            UI.confirmation_msg_false()
            pass
```

Getting value from UI entry boxes and storing them in local variables

Checking if the course entered is present in the course_data relation
If present : calls UI confirmation true() method which prints a “process successful” message and append name and course in student_data relation

If not present: calls UI confirmation false() method which prints an error message

Method for updating course_data relation



```python
@staticmethod
def update_c_table(course):
    course_name = course
    redundant_course = True
    for i in course_data["Course"]:
        if course_name == i:
            UI.confirmation_msg_repeat()
            break
        else:
            redundant_course = False
            pass

    if not redundant_course:
        course_data["Course"].append(course_name)
        course_df = pd.DataFrame(course_data)
        course_df.to_csv('course_data.csv', index=False)
        UI.confirmation_msg_true()
        pass
```

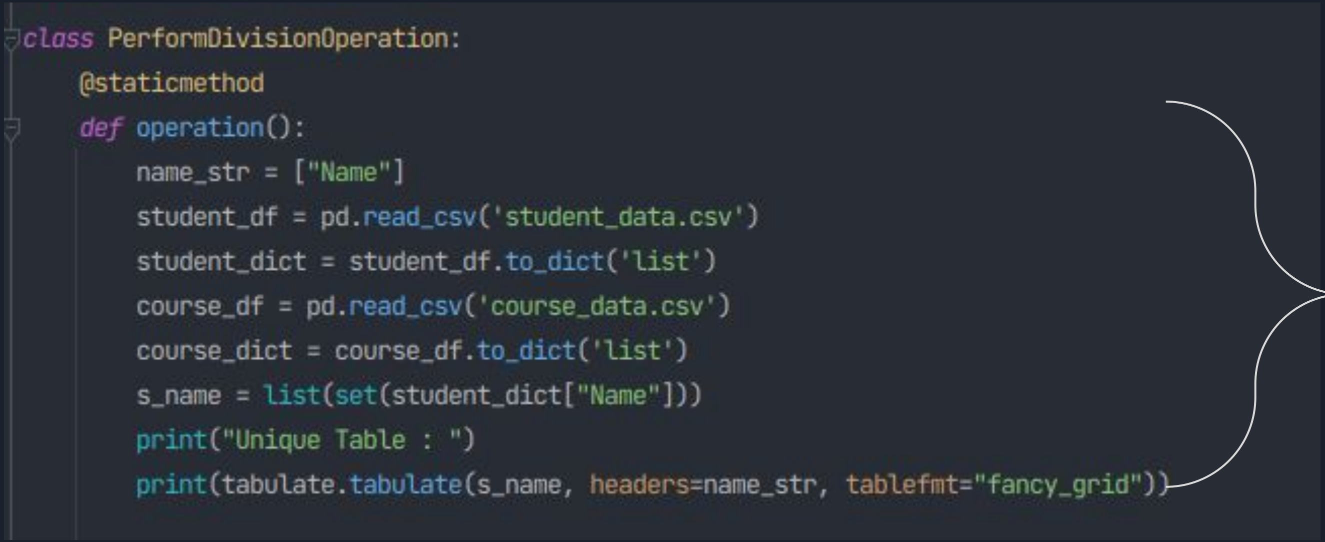
Getting course from UI entry box and storing it in a local variable

Checking if the course entered is already present in the course_data relation
If present: call for course repeat error

If not present: append the course into course_data relation and call "process successful" message

PerformDivisionOperation
Class

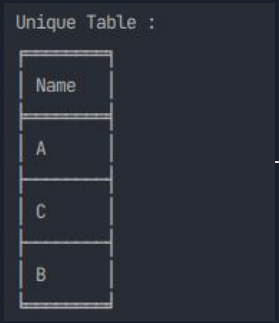
Method for performing division operator on student_data and course_data relation



```
class PerformDivisionOperation:
    @staticmethod
    def operation():
        name_str = ["Name"]
        student_df = pd.read_csv('student_data.csv')
        student_dict = student_df.to_dict('list')
        course_df = pd.read_csv('course_data.csv')
        course_dict = course_df.to_dict('list')
        s_name = list(set(student_dict["Name"]))
        print("Unique Table : ")
        print(tabulate.tabulate(s_name, headers=name_str, tablefmt="fancy_grid"))
```

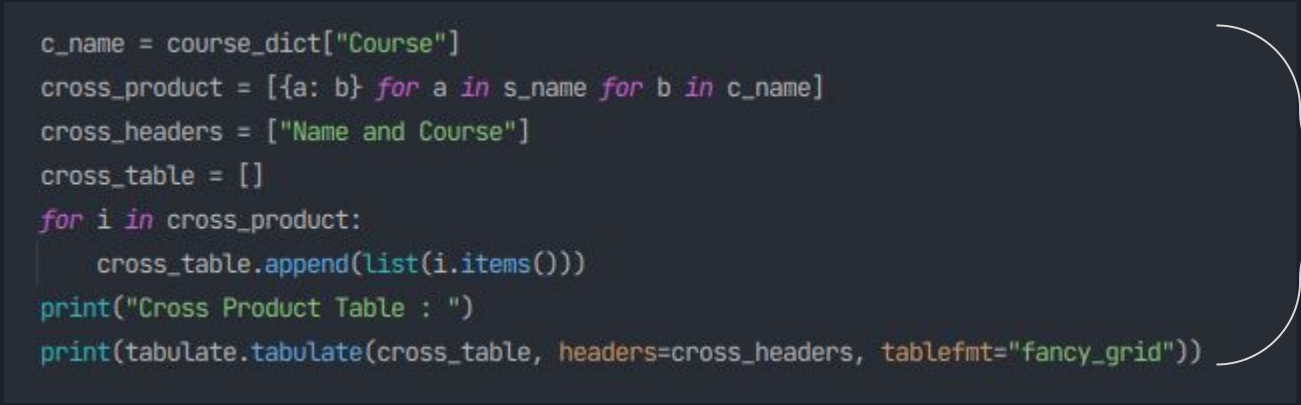
This snippet here prints the unique table by converting list of student names from student_data relation to set

set eliminates all repeating values, hence we get unique student names from student_data relation



```
Unique Table :
╒════════╕
│ Name   │
╞════════╡
│ A      │
├────────┤
│ C      │
├────────┤
│ B      │
╘════════╛
```

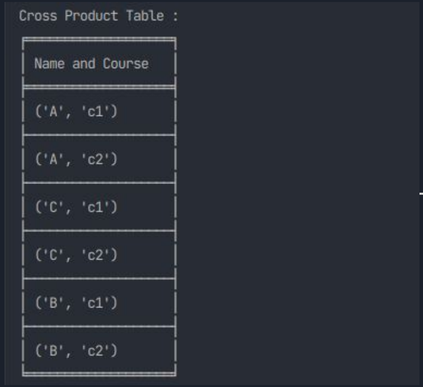
Output of this code snippet



```python
c_name = course_dict["Course"]
cross_product = [{a: b} for a in s_name for b in c_name]
cross_headers = ["Name and Course"]
cross_table = []
for i in cross_product:
    cross_table.append(list(i.items()))
print("Cross Product Table : ")
print(tabulate.tabulate(cross_table, headers=cross_headers, tablefmt="fancy_grid"))
```

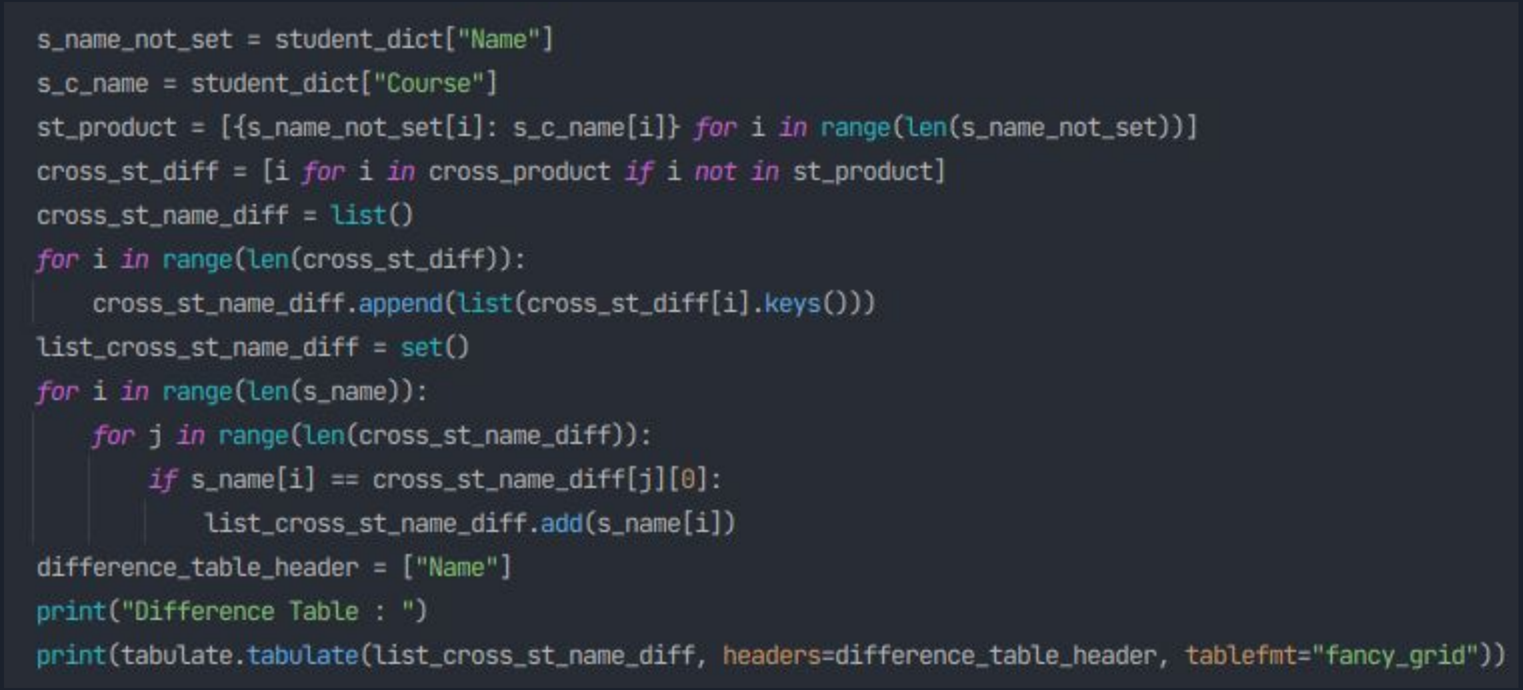
This snippet here prints out the cross product of student names from unique table and course from course_data relation

First, a list of dictionaries is created where the key is the student name and value is the course, and then each dictionary in the list is converted into a tuple for printing in tabular form.

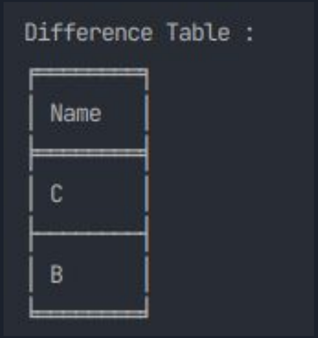


```
Cross Product Table :
╒═════════════════╕
│ Name and Course │
╞═════════════════╡
│ ('A', 'c1')     │
├─────────────────┤
│ ('A', 'c2')     │
├─────────────────┤
│ ('C', 'c1')     │
├─────────────────┤
│ ('C', 'c2')     │
├─────────────────┤
│ ('B', 'c1')     │
├─────────────────┤
│ ('B', 'c2')     │
╘═════════════════╛
```

Output of this code snippet

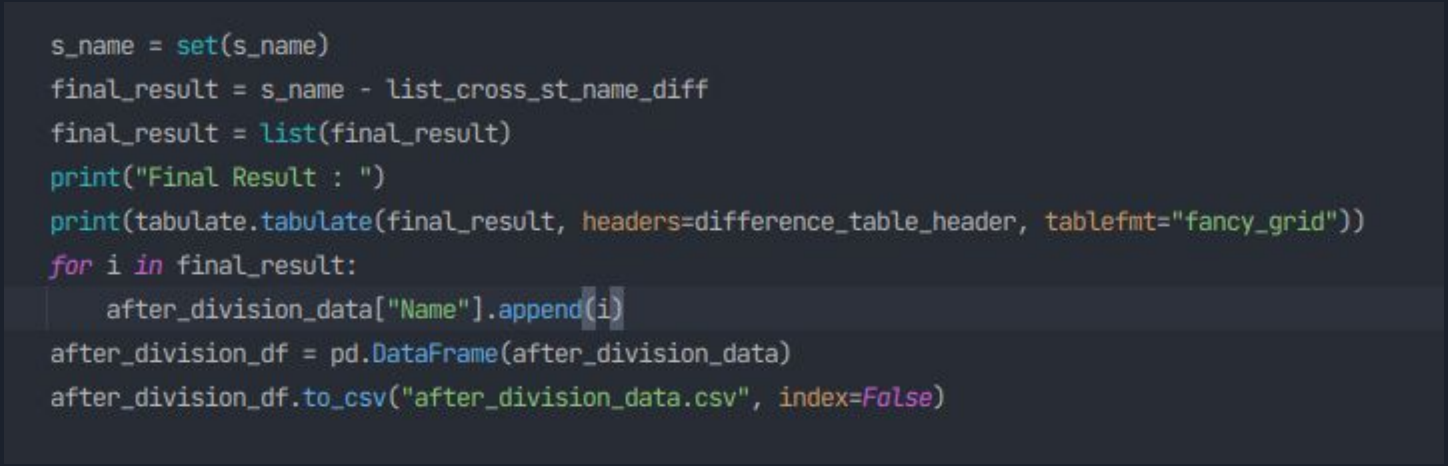


```
s_name_not_set = student_dict["Name"]
s_c_name = student_dict["Course"]
st_product = [{s_name_not_set[i]: s_c_name[i]} for i in range(len(s_name_not_set))]
cross_st_diff = [i for i in cross_product if i not in st_product]
cross_st_name_diff = list()
for i in range(len(cross_st_diff)):
    cross_st_name_diff.append(list(cross_st_diff[i].keys()))
list_cross_st_name_diff = set()
for i in range(len(s_name)):
    for j in range(len(cross_st_name_diff)):
        if s_name[i] == cross_st_name_diff[j][0]:
            list_cross_st_name_diff.add(s_name[i])
difference_table_header = ["Name"]
print("Difference Table : ")
print(tabulate.tabulate(list_cross_st_name_diff, headers=difference_table_header, tablefmt="fancy_grid"))
```

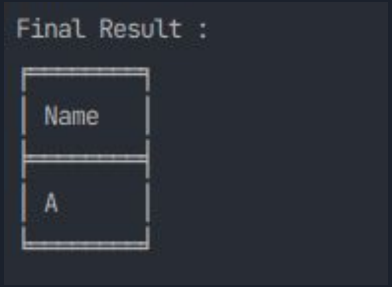


```
Difference Table :
╒══════╕
│ Name │
╞══════╡
│ C    │
├──────┤
│ B    │
╘══════╛
```

Here the difference table is determined by eliminating the values from cross table which are present in student_data relation, hence getting those student names which doesn’t corresponds to every course



```python
s_name = set(s_name)
final_result = s_name - list_cross_st_name_diff
final_result = list(final_result)
print("Final Result : ")
print(tabulate.tabulate(final_result, headers=difference_table_header, tablefmt="fancy_grid"))
for i in final_result:
    after_division_data["Name"].append(i)
after_division_df = pd.DataFrame(after_division_data)
after_division_df.to_csv("after_division_data.csv", index=False)
```



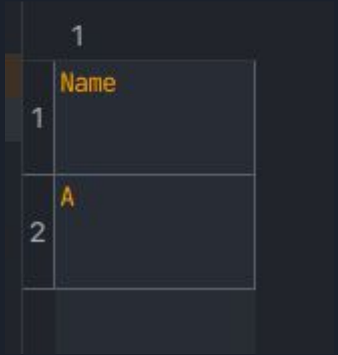
```
Final Result :

╒══════╕
│ Name │
╞══════╡
│ A    │
╘══════╛
```

In this snippet, we are eliminating values from the unique table which are present in difference table, hence getting those student names which corresponds to each and every course.

After doing so, we are appending the student names into after_division_data.csv so that we have an external record of the result.

This can also be considered as a backup result if somehow our python script get deleted or get filled with errors while adding a new feature.

After_division_data.csv :



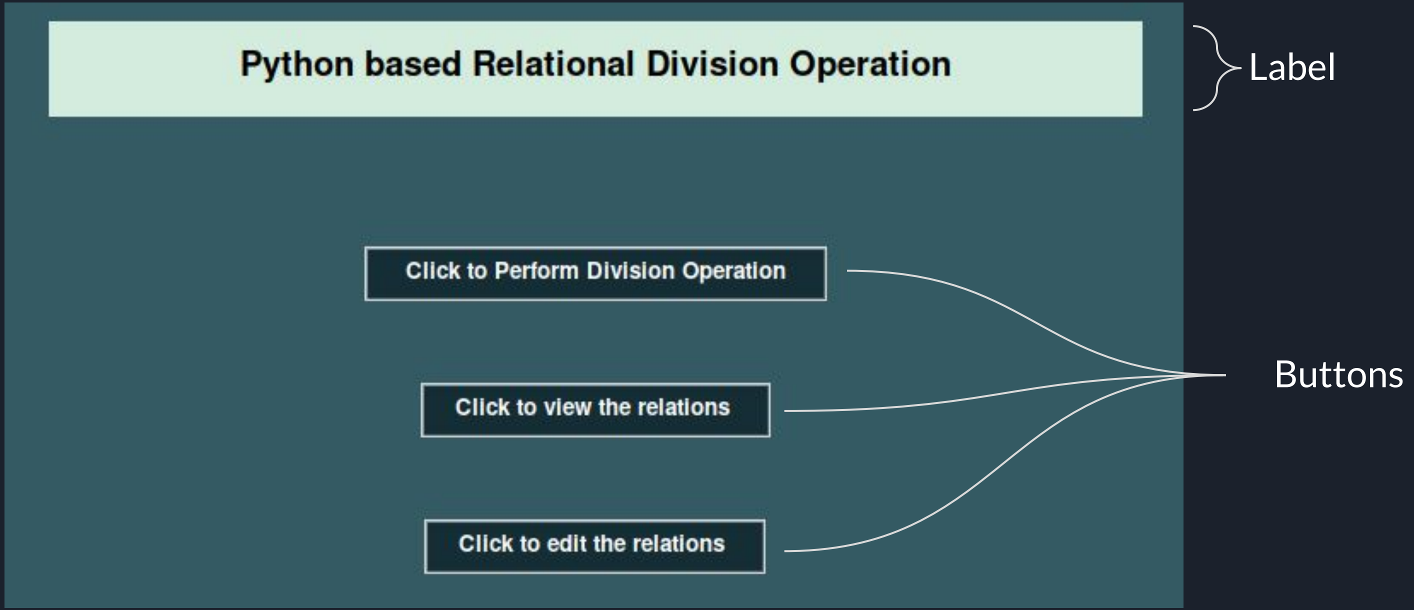
	1
1	Name
2	A

UI Class

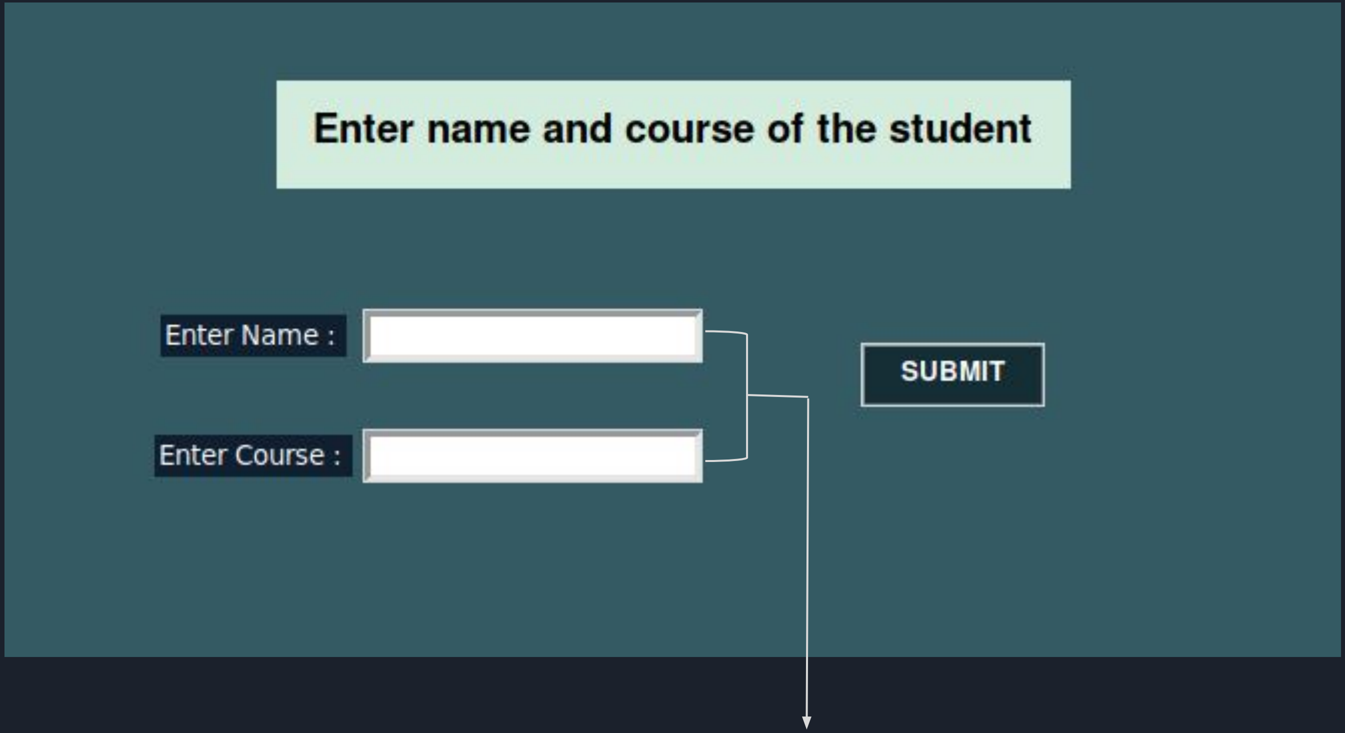
This class contains various methods that powers the GUI for this program.

This all includes : -

- Labels and Buttons :

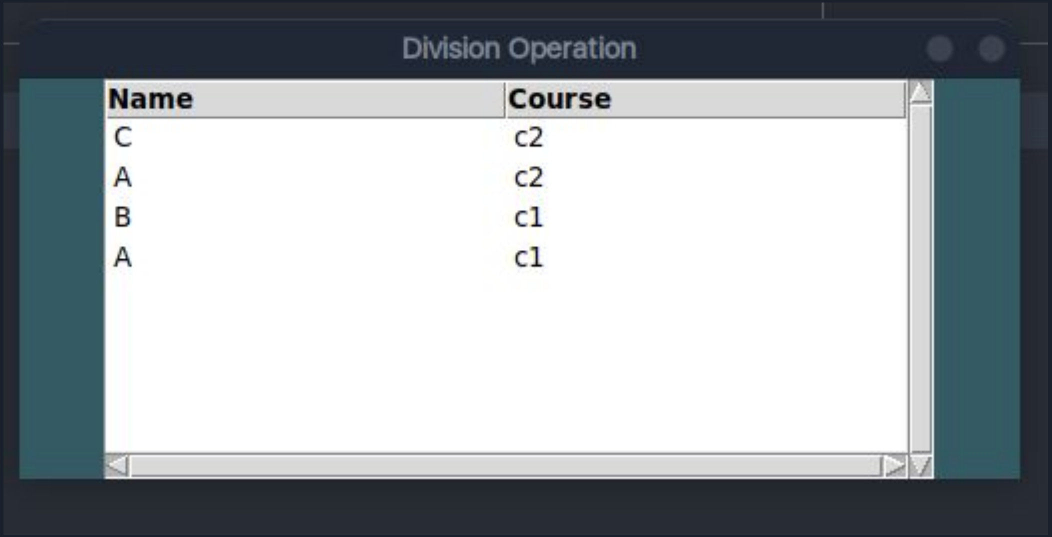

Python based Relational Division Operation
Label
Click to Perform Division Operation
Click to view the relations
Click to edit the relations
Buttons

- Entry Boxes :



Entry Boxes

- TreeView (ttk widget)


Division Operation

Name	Course
C	c2
A	c2
B	c1
A	c1

This tabular structure is known as TreeView

Features of this program :

- You can perform division operation on student_data and course_data and get final result in a treeview window, other tables in verbose mode.
- You can view the two relations from the program itself without manually going to the local directory and viewing the two CSVs.
- You can edit student_data.csv and course_data.csv from the program itself.

Thank

You !