import glob  
import os  
import random as rnd  
from random import shuffle  
import sqlite3 as sqlite  
import sys  
import shutil  
import prettytable as prettytable  
import xlsxwriter  
import winsound  
import sys, string  
import subprocess  
  
import tkinter  
from tkinter import \*  
import tkinter.ttk as ttk  
from tkinter import messagebox  
from enum import Enum  
import multiprocessing  
from multiprocessing import Process  
import sys, platform  
from subprocess import PIPE, Popen, STDOUT  
import sys, string  
import threading  
import re  
import time  
  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import csv  
import warnings  
import statsmodels  
import statsmodels.tsa.statespace.\_filters  
import statsmodels.tsa.statespace.\_filters.\_conventional  
import statsmodels.tsa.statespace.\_filters.\_univariate  
import statsmodels.tsa.statespace.\_filters.\_univariate\_diffuse  
import statsmodels.tsa.statespace.\_filters.\_inversions  
import statsmodels.tsa.statespace.\_smoothers  
import statsmodels.tsa.statespace.\_smoothers.\_conventional  
import statsmodels.tsa.statespace.\_smoothers.\_univariate  
import statsmodels.tsa.statespace.\_smoothers.\_univariate\_diffuse  
import statsmodels.tsa.statespace.\_smoothers.\_classical  
import statsmodels.tsa.statespace.\_smoothers.\_alternative  
from statsmodels.\_\_init\_\_ import \_\_version\_\_  
import sklearn  
import sklearn.utils.\_weight\_vector  
from pmdarima.arima import auto\_arima  
from pmdarima.arima import ADFTest  
from pmdarima.arima import PPTest  
from sklearn.metrics import r2\_score  
warnings.filterwarnings(**'ignore'**)  
  
def ECESPV():  
 SETTING\_INFO = **'C:/Program Files/ECE-SPV/settinginfo.txt'** OUTPUT\_INFO = **'C:/Program Files/ECE-SPV/outputinfo.txt'** ARASTDIN = **'C:/Program Files/ECE-SPV/arastdin.txt'** ARASTDOUT = **'C:/Program Files/ECE-SPV/arastdout.txt'** restorePoint=sys.stdout  
 class FullScreenApp(object):  
 def \_\_init\_\_(self, master, \*\*kwargs):  
 self.master=master  
 pad=3  
 self.\_geom=**'200x200+0+0'** master.geometry(**"{0}x{1}+0+0"**.format(  
 master.winfo\_screenwidth() - pad, master.winfo\_screenheight() - pad))  
 master.bind(**'<Escape>'**, self.toggle\_geom)  
  
 def toggle\_geom(self, event):  
 geom=self.master.winfo\_geometry()  
 self.master.geometry(self.\_geom)  
 self.\_geom=geom  
 class StdoutRedirector(object):  
  
 def \_\_init\_\_(self, text\_area):  
 self.text\_area = text\_area  
  
 def write(self, str):  
 self.text\_area.insert(END, str)  
 self.text\_area.see(END)  
 root=Tk()  
 root.title(**"ECE-SPV"**)  
 root.wm\_iconbitmap(**"C:/Program Files/ECE-SPV/ece-spv.ico"**)  
 screen\_width=root.winfo\_screenwidth()  
 screen\_height=root.winfo\_screenheight()  
 width=900  
 height=627  
 x=(screen\_width / 2) - (width / 2)  
 y=(screen\_height / 2) - (height / 2)  
 root.geometry(**'%dx%d+%d+%d'** % (width, height, x, y))  
 root.resizable(1, 1)  
  
 *# ==================================COMMANDS============================================* def ARADB():  
 global conn, cursor  
 conn=sqlite.connect(**'ARADB.db'**)  
 cursor=conn.cursor()  
  
  
 def Init\_ARADB():  
 global conn, cursor  
 ARADB()  
 cursor.execute(**"CREATE TABLE IF NOT EXISTS `dept` (code TEXT PRIMARY KEY)"**)  
 cursor.execute(**"CREATE TABLE IF NOT EXISTS `curriculum` (code TEXT PRIMARY KEY)"**)  
 cursor.execute(**"CREATE TABLE IF NOT EXISTS `section` (code TEXT PRIMARY KEY)"**)  
 cursor.execute(**"CREATE TABLE IF NOT EXISTS `section\_subject` (section\_code TEXT NOT NULL, subject\_code TEXT NOT NULL)"**)  
 cursor.execute(**"CREATE TABLE IF NOT EXISTS `instructor` (number TEXT PRIMARY KEY, name TEXT NOT NULL, max\_hours INTEGER NOT NULL, seniority INTEGER NOT NULL)"**)  
 cursor.execute(**"CREATE TABLE IF NOT EXISTS `subject` (code TEXT PRIMARY KEY, description TEXT, units INTEGER NOT NULL, numbHour INTEGER NOT NULL, compatibility TEXT NOT NULL, curriculum TEXT NOT NULL, max\_numb\_of\_students INTEGER NOT NULL, dept TEXT NOT NULL)"**)  
 cursor.execute(**"CREATE TABLE IF NOT EXISTS `room` (number TEXT PRIMARY KEY, capacity INTEGER NOT NULL, type TEXT NOT NULL)"**)  
 cursor.execute(**"CREATE TABLE IF NOT EXISTS `section\_availability` (section\_code TEXT NOT NULL, meeting\_time\_id TEXT NOT NULL, start TEXT, end TEXT)"**)  
 cursor.execute(**"CREATE TABLE IF NOT EXISTS `instructor\_availability` (instructor\_id TEXT NOT NULL, meeting\_time TEXT NOT NULL, start TEXT, end TEXT)"**)  
 cursor.execute(**"CREATE TABLE IF NOT EXISTS `instructor\_overtime` (instructor\_id TEXT NOT NULL, meeting\_time TEXT NOT NULL, start TEXT, end TEXT)"**)  
 cursor.execute(**"CREATE TABLE IF NOT EXISTS `subject\_instructor` (subject\_code TEXT NOT NULL, instructor\_number TEXT NOT NULL)"**)  
 cursor.execute(**"CREATE TABLE IF NOT EXISTS `room\_availability` (room\_number TEXT NOT NULL, meeting\_time\_id TEXT NOT NULL, start TEXT, end TEXT)"**)  
 cursor.execute(**"CREATE TABLE IF NOT EXISTS `population` (semester TEXT PRIMARY KEY, number INTEGER NOT NULL)"**)  
 cursor.close()  
 conn.close()  
  
  
 Init\_ARADB()  
  
  
 def combobox\_input():  
 dst=**"C:/Program Files/ECE-SPV/DB Saved"** dstfolder=os.listdir(dst)  
 number=[]  
 for files in dstfolder:  
 number.append(files)  
 return number  
  
  
 def Save():  
 SV=Toplevel(root)  
 SV.title(**"ECE-SPV"**)  
 SV.wm\_iconbitmap(**"ece-spv.ico"**)  
 SV.screen\_width=SV.winfo\_screenwidth()  
 SV.screen\_height=SV.winfo\_screenheight()  
 SV.geometry(**'%dx%d+%d+%d'** % (300, 160, (screen\_width / 2) - (width / 2), (screen\_height / 2) - (height / 2)))  
 SV.resizable(0, 0)  
  
 def Confirm():  
 if DB.get() == **""**:  
 txt\_result1.config(text = **"Please enter a database name!"**, fg = **"red"**)  
 else:  
 src=**"C:/Program Files/ECE-SPV"** dst=**"C:/Program Files/ECE-SPV/DB Saved"** dstfolder=os.listdir(dst)  
 if (str(DB.get()) + **".db"**) not in dstfolder:  
 shutil.copyfile(src = src + **"/ARADB.db"**, dst = dst + **"/"** + str(DB.get()) + **".db"**)  
 SV.destroy()  
 View()  
 txt\_result.config(text = **"Database is saved successfully!"**, fg = **"green"**)  
 else:  
 txt\_result1.config(text = **"Database name already exists!"**, fg = **"red"**)  
  
 frame=Frame(SV, relief = **"raise"**)  
 frame.pack()  
 txt\_filename=Label(frame, text = **"Enter a database name:"**, font = (**'arial'**, 13), bd = 15)  
 txt\_filename.pack()  
 e\_filename=Entry(frame, textvariable = DB, width = 41)  
 e\_filename.pack()  
 e\_filename.delete(0, END)  
 Buttons=Frame(frame, bd = 8, relief = **"raise"**)  
 Buttons.pack(side=BOTTOM, pady=10)  
 txt\_result1=Label(Buttons)  
 txt\_result1.pack()  
 btn\_confirm=Button(Buttons, width = 28, text = **"Confirm"**, font = (**'arial'**, 10, **'bold'**),  
 fg = **"Black"**, command = Confirm)  
 btn\_confirm.pack()  
  
 if \_\_name\_\_ == **'\_\_main\_\_'**:  
 SV.mainloop()  
  
  
 def Load():  
 if DB.get() == **""**:  
 txt\_result.config(text = **"Please select or enter a Database!"**, fg = **"red"**)  
 else:  
 src=**"C:/Program Files/ECE-SPV/DB Saved"** dst=**"C:/Program Files/ECE-SPV"** shutil.copyfile(src = src + **"/"** + str(DB.get()), dst = dst + **"/ARADB.db"**)  
 cb\_db.delete(0, **'end'**)  
 View()  
 txt\_result.config(text = **"Database is loaded successfully."**, fg = **"green"**)  
  
  
 def Delete():  
 if DB.get() == **""**:  
 txt\_result.config(text = **"Please select or enter a Database!"**, fg = **"red"**)  
 else:  
 result=messagebox.askquestion(**"Delete"**, **"Do you want to delete selected Database?"**, icon = **'warning'**)  
 if result == **'yes'**:  
 dst=**"C:/Program Files/ECE-SPV/DB Saved"** os.remove(dst + **"/"** + str(DB.get()))  
 cb\_db.delete(0, **'end'**)  
 View()  
 txt\_result.config(text = **"Database is deleted Successfully!"**, fg = **"green"**)  
 else:  
 pass  
  
  
 def Update():  
 try:  
 with open(ARASTDOUT,**"r"**) as f:  
 OUTPUT=f.read()  
 textout.insert(**"1.0"**, OUTPUT)  
 f.close()  
 except IOError:  
 with open(ARASTDOUT,**"w"**) as f:  
 f.write(**''**)  
 f.close()  
  
 a=textout.yview()[0]  
 b=textout.yview()[1]  
 textout.yview\_moveto(str(1-a))  
  
  
 def View():  
 tree.delete(\*tree.get\_children())  
 dst=**"C:/Program Files/ECE-SPV/DB Saved"** dstfolder=os.listdir(dst)  
 number=list()  
 for files in dstfolder:  
 tree.insert(**''**, **'end'**, values = ((len(number) + 1), files))  
 number.append(files)  
 txt\_result.config(text = **"Successfully viewed from DB Saved Folder"**, fg = **"Black"**)  
  
  
 def ClearSTD():  
 textout.delete(**"1.0"**, END) *# if you want to remove the old data* with open(ARASTDOUT, **'w'**) as f:  
 f.write(**''**)  
 f.close()  
 with open(ARASTDIN, **'w'**) as f:  
 f.write(**''**)  
 f.close()  
  
  
 def click(event):  
 try:  
 selected=tree.focus() *# Grab record position/number* values=tree.item(selected, **'values'**) *# Grab record values* cb\_db.delete(0, **'end'**)  
 cb\_db.insert(0, values[1])  
 except IndexError: pass  
  
  
 def enter(event):  
 *# Get text from textin* INPUT=TEXTIN.get()+**'**\n**'** sys.stdin.write(INPUT.rstrip(**'**\n**'**))  
 sys.stdout.write(INPUT)  
 *# Display the last line of text from textin to ARASTDOUT file* with open(ARASTDOUT, **'a+'**) as f:  
 f.write(INPUT)  
 f.close()  
 textout.insert(**'end'**, INPUT, **"1.0"**)  
 TEXTIN.set(**''**)  
  
  
 def reset\_scrollregion(event):  
 textout.configure(scrollregion=textout.bbox(**"all"**))  
  
  
 def Add\_Section():  
 SC=Toplevel(root)  
 SC.title(**"ECE-SPV"**)  
 SC.wm\_iconbitmap(**"ece-spv.ico"**)  
 SC.screen\_width=SC.winfo\_screenwidth()  
 SC.screen\_height=SC.winfo\_screenheight()  
 SC.width=720  
 SC.height=627  
 SC.x=(screen\_width / 2) - (width / 2)  
 SC.y=(screen\_height / 2) - (height / 2)  
 SC.geometry(**'%dx%d+%d+%d'** % (width, height, x, y))  
 SC.resizable(0, 0)  
  
 *# ==================================COMMANDS============================================* def click(event):  
 try:  
 selected=tree.focus() *# Grab record position/number* values=tree.item(selected, **'values'**) *# Grab record values* cb\_code.delete(0, **'end'**)  
 cb\_code.insert(0, values[0])  
 except IndexError: pass  
  
 def combobox\_input1():  
 ARADB()  
 cursor.execute(**"SELECT code FROM `section`"**) *# SELECT code FROM `physical\_room`* conn.commit()  
 data=[]  
 for row in cursor.fetchall():  
 data.append(row[0])  
 return data  
 cursor.close()  
 conn.close()  
  
 def Create():  
 if CODE.get() == **""**:  
 txt\_result.config(text = **"Please enter a Section Code!"**, fg = **"red"**)  
 else:  
 ARADB()  
 cursor.execute(**"INSERT INTO `section` (code) VALUES(?)"**, (str(CODE.get()),))  
 conn.commit()  
 CODE.set(**""**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Created a data!"**, fg = **"green"**)  
  
 def View():  
 tree.delete(\*tree.get\_children())  
 ARADB()  
 cursor.execute(**"SELECT \* FROM `section` ORDER BY `code` ASC"**)  
 fetch=cursor.fetchall()  
 for data in fetch:  
 tree.insert(**''**, **'end'**, values = (data[0]))  
 cursor.close()  
 conn.close()  
 txt\_result.config(text = **"Successfully viewed the data from database"**, fg = **"black"**)  
  
 def Delete():  
 if CODE.get() == **""**:  
 txt\_result.config(text = **"Please enter a Section Code!"**, fg = **"red"**)  
 else:  
 ARADB()  
 cursor.execute(**"DELETE FROM 'section' WHERE code = '"** + str(CODE.get()) + **"'"**)  
 conn.commit()  
 cb\_code.delete(0, **'end'**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Deleted Successfully!"**, fg = **"green"**)  
  
 def Clear():  
 result=messagebox.askquestion(**"Clear"**, **"This will clear the Section table.**\n**Do you want to proceed?"**, icon = **'warning'**)  
 if result == **'yes'**:  
 ARADB()  
 cursor.execute(**"""DROP TABLE section"""**)  
 cursor.execute(**"""create table section (code text PRIMARY KEY)"""**)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Table Cleared Successfully!"**, fg = **"green"**)  
 else:  
 pass  
  
 def Modify\_Availability():  
 SA=Toplevel(SC)  
 SA.title(**"ECE-SPV"**)  
 SA.wm\_iconbitmap(**"ece-spv.ico"**)  
 SA.screen\_width=SA.winfo\_screenwidth()  
 SA.screen\_height=SA.winfo\_screenheight()  
 SA.width=720  
 SA.height=627  
 SA.x=(screen\_width / 2) - (width / 2)  
 SA.y=(screen\_height / 2) - (height / 2)  
 SA.geometry(**'%dx%d+%d+%d'** % (width, height, x, y))  
 SA.resizable(0, 0)  
  
 *# ==================================COMMANDS============================================* class DBMgr:  
 def \_\_init\_\_(self):  
 self.\_conn=sqlite.connect(**'ARADB.db'**) *# sql connector* self.\_cursor=self.\_conn.cursor() *# sql cursor* self.\_times=self.\_select\_times() *# select time sql cursor* self.\_timepoints=self.\_select\_timepoints() *# select timepoints sql cursor* self.\_meetingTimes=self.\_select\_meeting\_times() *# select meeting time sql cursor* self.\_instructors=self.\_select\_instructors() *# select instructor ql cursor* self.\_subjects=self.\_select\_subjects() *# select subject sql cursor* self.\_rooms=self.\_select\_rooms() *# select room sql cursor* self.\_sections=self.\_select\_sections() *# select section sql cursor* self.\_numberOfClasses=0 *# initial number of classes* for i in range(0, len(self.\_sections)):  
 self.\_numberOfClasses+=len(  
 self.\_sections[i].get\_subjects()) *# auto-itereation of number of classes creation  
  
 # select section sql command* def \_select\_sections(self):  
 self.\_cursor.execute(**"SELECT \* FROM section"**)  
 sections=self.\_cursor.fetchall()  
 returnSections=[]  
 for i in range(0, len(sections)):  
 returnSections.append(Section(sections[i][0], self.\_select\_section\_subjects(sections[i][0]),  
 self.\_select\_section\_availability(sections[i][0])))  
 return returnSections  
  
 *# select subject sql command* def \_select\_subjects(self):  
 self.\_cursor.execute(**"SELECT \* FROM subject"**)  
 subjects=self.\_cursor.fetchall()  
 returnSubjects=[]  
 for i in range(0, len(subjects)):  
 returnSubjects.append(  
 Subject(subjects[i][0], subjects[i][1], subjects[i][2], subjects[i][3], subjects[i][4],  
 subjects[i][5], subjects[i][6], subjects[i][7], self.\_select\_subject\_instructors(  
 subjects[i][0])))  
 return returnSubjects  
  
 *# select instructor sql command* def \_select\_instructors(self):  
 self.\_cursor.execute(**"SELECT \* FROM instructor"**)  
 instructors=self.\_cursor.fetchall()  
 returnInstructors=[]  
 for i in range(0, len(instructors)):  
 returnInstructors.append(  
 Instructor(instructors[i][0], instructors[i][1], instructors[i][2], instructors[i][3],  
 self.\_select\_instructor\_availability(instructors[i][0]),  
 self.\_select\_instructor\_availability\_start(instructors[i][0]),  
 self.\_select\_instructor\_availability\_end(instructors[i][0]),  
 self.\_select\_instructor\_overtime(instructors[i][0])))  
 return returnInstructors  
  
 *# select room sql command* def \_select\_rooms(self):  
 self.\_cursor.execute(**"SELECT \* FROM room"**)  
 rooms=self.\_cursor.fetchall()  
 returnRooms=[]  
 for i in range(0, len(rooms)):  
 returnRooms.append(Room(rooms[i][0], rooms[i][1], rooms[i][2], rooms[i][3],  
 self.\_select\_room\_availability(  
 rooms[i][  
 0])))  
 return returnRooms  
  
 *# select meeting time sql command* def \_select\_meeting\_times(self):  
 self.\_cursor.execute(**"SELECT \* FROM meeting\_time"**)  
 meetingTimes=self.\_cursor.fetchall()  
 returnMeetingTimes=[]  
 for i in range(0, len(meetingTimes)):  
 returnMeetingTimes.append(  
 MeetingTime(meetingTimes[i][0], meetingTimes[i][1], meetingTimes[i][2],  
 meetingTimes[i][3], meetingTimes[i][4],  
 self.\_select\_rest\_time(meetingTimes[i][0]),  
 self.\_select\_break\_time(meetingTimes[i][0])))  
 return returnMeetingTimes  
  
 *# select time sql command* def \_select\_times(self):  
 self.\_cursor.execute(**"SELECT \* FROM time"**)  
 times=self.\_cursor.fetchall()  
 returnTimes=[]  
 for i in range(0, len(times)):  
 returnTimes.append(Time(times[i][0]))  
 return returnTimes  
  
 *# select timepoint sql command* def \_select\_timepoints(self):  
 self.\_cursor.execute(**"SELECT \* FROM time\_point"**)  
 timepoints=self.\_cursor.fetchall()  
 returnTimePoints=[]  
 for i in range(0, len(timepoints)):  
 returnTimePoints.append(TimePoint(timepoints[i][0],  
 self.\_select\_point\_block(timepoints[i][0]),  
 self.\_select\_phantom\_block(timepoints[i][0])))  
 return returnTimePoints  
  
 *# GETTING VALUE* def \_select\_section\_subjects(self, sectionCode):  
 self.\_cursor.execute(**"SELECT \* FROM section\_subject where section\_code == '"** + sectionCode + **"'"**)  
 dbSubjectCodes=self.\_cursor.fetchall()  
 subjectCodes=[]  
 for i in range(0, len(dbSubjectCodes)):  
 subjectCodes.append(dbSubjectCodes[i][1])  
 returnValue=[]  
 for i in range(0, len(self.\_subjects)):  
 if self.\_subjects[i].get\_code() in subjectCodes:  
 returnValue.append(self.\_subjects[i])  
 return returnValue  
  
 def \_select\_subject\_instructors(self, subjectCode):  
 self.\_cursor.execute(**"SELECT \* FROM subject\_instructor where subject\_code == '"** + subjectCode + **"'"**)  
 dbInstructorNumbers=self.\_cursor.fetchall()  
 instructorNumbers=[]  
 for i in range(0, len(dbInstructorNumbers)): instructorNumbers.append(dbInstructorNumbers[i][1])  
 returnValue=[]  
 for i in range(0, len(self.\_instructors)):  
 if self.\_instructors[i].get\_number() in instructorNumbers:  
 returnValue.append(self.\_instructors[i])  
 return returnValue  
  
 def \_select\_instructor\_availability(self, instructor):  
 self.\_cursor.execute(  
 **"SELECT \* from instructor\_availability where instructor\_id == '"** + instructor + **"'"**)  
 instructorMTsRS=self.\_cursor.fetchall()  
 instructorMTs=[]  
 for i in range(0, len(instructorMTsRS)):  
 instructorMTs.append(instructorMTsRS[i][1])  
 instructorAvailability=list()  
 for i in range(0, len(self.\_meetingTimes)):  
 if self.\_meetingTimes[i].get\_id() in instructorMTs:  
 instructorAvailability.append(self.\_meetingTimes[i].get\_id())  
 return instructorAvailability  
  
 def \_select\_instructor\_availability\_start(self, instructor):  
 self.\_cursor.execute(  
 **"SELECT \* from instructor\_availability where instructor\_id == '"** + instructor + **"'"**)  
 instructorMTsRS=self.\_cursor.fetchall()  
 instructorMTsSTART=[]  
 for i in range(0, len(instructorMTsRS)):  
 instructorMTsSTART.append(instructorMTsRS[i][2])  
 instructorAvailabilitySTART=list()  
 for i in range(0, len(self.\_timepoints)):  
 if self.\_timepoints[i].get\_point() in instructorMTsSTART:  
 instructorAvailabilitySTART.append(self.\_timepoints[i].get\_point())  
 return instructorAvailabilitySTART  
  
 def \_select\_instructor\_availability\_end(self, instructor):  
 self.\_cursor.execute(  
 **"SELECT \* from instructor\_availability where instructor\_id == '"** + instructor + **"'"**)  
 instructorMTsRS=self.\_cursor.fetchall()  
 instructorMTsEND=[]  
 for i in range(0, len(instructorMTsRS)):  
 instructorMTsEND.append(instructorMTsRS[i][3])  
 instructorAvailabilityEND=list()  
 for i in range(0, len(self.\_timepoints)):  
 if self.\_timepoints[i].get\_point() in instructorMTsEND:  
 instructorAvailabilityEND.append(self.\_timepoints[i].get\_point())  
 return instructorAvailabilityEND  
  
 def \_select\_instructor\_overtime(self, instructor):  
 self.\_cursor.execute(  
 **"SELECT \* from instructor\_overtime where instructor\_id == '"** + instructor + **"'"**)  
 instructorMTsRS=self.\_cursor.fetchall()  
 instructorMTs=[]  
 for i in range(0, len(instructorMTsRS)):  
 instructorMTs.append(instructorMTsRS[i][1])  
 instructorOvertime=list()  
 for i in range(0, len(self.\_meetingTimes)):  
 if self.\_meetingTimes[i].get\_id() in instructorMTs:  
 instructorOvertime.append(self.\_meetingTimes[i].get\_id())  
 return instructorOvertime  
  
 def \_select\_room\_availability(self, room):  
 self.\_cursor.execute(**"SELECT \* from room\_availability where room\_number = '"** + room + **"'"**)  
 roomMTsRS=self.\_cursor.fetchall()  
 roomMTs=[]  
 for i in range(0, len(roomMTsRS)): roomMTs.append(roomMTsRS[i][1])  
 roomAvailability=list()  
 for i in range(0, len(self.\_meetingTimes)):  
 if self.\_meetingTimes[i].get\_id() in roomMTs:  
 roomAvailability.append(self.\_meetingTimes[i].get\_id())  
 return roomAvailability  
  
 def \_select\_section\_availability(self, section):  
 self.\_cursor.execute(**"SELECT \* from section\_availability where section\_code = '"** + section + **"'"**)  
 sectionMTsRS=self.\_cursor.fetchall()  
 sectionMTs=[]  
 for i in range(0, len(sectionMTsRS)): sectionMTs.append(sectionMTsRS[i][1])  
 sectionAvailability=list()  
 for i in range(0, len(self.\_meetingTimes)):  
 if self.\_meetingTimes[i].get\_id() in sectionMTs:  
 sectionAvailability.append(self.\_meetingTimes[i].get\_id())  
 return sectionAvailability  
  
 def \_select\_rest\_time(self, meeting\_time):  
 self.\_cursor.execute(**"SELECT \* from rest\_time where meeting\_time\_id = '"** + meeting\_time + **"'"**)  
 timeMTsRS=self.\_cursor.fetchall()  
 timeMTs=[]  
 for i in range(0, len(timeMTsRS)): timeMTs.append(timeMTsRS[i][1])  
 restTime=list()  
 for i in range(0, len(self.\_times)):  
 if self.\_times[i].get\_block() in timeMTs:  
 restTime.append(self.\_times[i].get\_block())  
 return restTime  
  
 def \_select\_break\_time(self, meeting\_time):  
 self.\_cursor.execute(**"SELECT \* from break\_time where meeting\_time\_id = '"** + meeting\_time + **"'"**)  
 timeMTsRS=self.\_cursor.fetchall()  
 timeMTs=[]  
 for i in range(0, len(timeMTsRS)): timeMTs.append(timeMTsRS[i][1])  
 breakTime=list()  
 for i in range(0, len(self.\_times)):  
 if self.\_times[i].get\_block() in timeMTs:  
 breakTime.append(self.\_times[i].get\_block())  
 return breakTime  
  
 def \_select\_point\_block(self, point):  
 self.\_cursor.execute(**"SELECT \* from point\_block where time\_point == '"** + point + **"'"**)  
 pointBlocksRS=self.\_cursor.fetchall()  
 pointBlocks=[]  
 for i in range(0, len(pointBlocksRS)): pointBlocks.append(pointBlocksRS[i][1])  
 returnValue=list()  
 for i in range(0, len(self.\_times)):  
 if self.\_times[i].get\_block() in pointBlocks:  
 returnValue.append(self.\_times[i].get\_block())  
 return returnValue  
  
 def \_select\_phantom\_block(self, point):  
 self.\_cursor.execute(**"SELECT \* from phantom\_block where time\_point == '"** + point + **"'"**)  
 pointBlocksRS=self.\_cursor.fetchall()  
 pointBlocks=[]  
 for i in range(0, len(pointBlocksRS)): pointBlocks.append(pointBlocksRS[i][1])  
 returnValue=list()  
 for i in range(0, len(self.\_times)):  
 if self.\_times[i].get\_block() in pointBlocks:  
 returnValue.append(self.\_times[i].get\_block())  
 return returnValue  
  
 def get\_sections(self):  
 return self.\_sections *# get item to section table* def get\_subjects(self):  
 return self.\_subjects *# get item to subject table* def get\_instructors(self):  
 return self.\_instructors *# get item to instructor table* def get\_meetingTimes(self):  
 return self.\_meetingTimes *# get item to meeting time table* def get\_times(self):  
 return self.\_times *# get item to time table* def get\_timepoints(self):  
 return self.\_timepoints *# get item to time table* def get\_rooms(self):  
 return self.\_rooms *# get item to room table* def get\_numberOfClasses(self):  
 return self.\_numberOfClasses *# get item to number of class table* class Schedule:  
 def \_\_init\_\_(self):  
 self.\_data=dbMgr  
 self.\_classes=[]  
 self.\_conflicts=[]  
 self.\_fitness=-1  
 self.\_classNumb=0  
 self.\_isFitnessChanged=True  
 self.\_generationNumber=0  
  
 def get\_classes(self):  
 self.\_isFitnessChanged=True  
 return self.\_classes  
  
 def get\_conflicts(self):  
 return self.\_conflicts  
  
 def get\_fitness(self):  
 if (self.\_isFitnessChanged == True):  
 self.\_fitness=self.calculate\_fitness()  
 self.\_isFitnessChanged=False  
 return self.\_fitness  
  
 def initialize(self):  
 sections=dbMgr.get\_sections() *# point to section table* meetingTimes=dbMgr.get\_meetingTimes()  
 rooms=dbMgr.get\_rooms()  
 classes=self.\_classes  
 PR=list()  
 instructorList=list()  
 MT1=list()  
 MT1p5=list()  
 MT2=list()  
 MT3=list()  
 MT4=list()  
 MT1v2=list()  
 MT1p5v2=list()  
 MT2v2=list()  
 MT3v2=list()  
 for RM in range(0, len(rooms)):  
 if (rooms[RM].get\_type() == **'PR'**):  
 PR.append(rooms[RM])  
 for MT in range(0, len(meetingTimes)):  
 if (meetingTimes[MT].get\_MThour() == 1):  
 MT1.append(meetingTimes[MT])  
 MT1v2.append(meetingTimes[MT])  
 if (meetingTimes[MT].get\_MThour() == 1.5):  
 MT1p5.append(meetingTimes[MT])  
 MT1p5v2.append(meetingTimes[MT])  
 if (meetingTimes[MT].get\_MThour() == 2):  
 MT2.append(meetingTimes[MT])  
 MT2v2.append(meetingTimes[MT])  
 if (meetingTimes[MT].get\_MThour() == 3):  
 MT3.append(meetingTimes[MT])  
 MT3v2.append(meetingTimes[MT])  
 if (meetingTimes[MT].get\_MThour() == 4):  
 MT4.append(meetingTimes[MT])  
 for i in range(0, len(sections)): *# List all the items per coloumn* subjects=sections[i].get\_subjects() *# point to subject table* for j in range(0, len(subjects)): *# List all the items per coloumn* newClass=Class(self.\_classNumb, sections[i], subjects[j]) *# Create a new class* self.\_classNumb+=1 *# Iterate the class creation* newClass.set\_instructor(subjects[j].get\_instructors()[rnd.randrange(0, len(subjects[  
 j].get\_instructors()))]) *# Pick a random instructor taht can teach the assigned subject for the new class* if (subjects[j].get\_compatibility() == **'PR'**):  
 newClass.set\_room(  
 PR[rnd.randrange(0, len(PR))]) *# Pick a random room in database for the new class* else:  
 newClass.set\_room(  
 rooms[rnd.randrange(0,  
 len(rooms))]) *# Pick a random room in database for the new class"""* def wholetwohours():  
 newClass.set\_meetingTime(MT2[rnd.randrange(0, len(MT2))])  
 newClass.set\_meetingTime1(dbMgr.get\_meetingTimes()[0])  
  
 def wholethreehours():  
 newClass.set\_meetingTime(MT3[rnd.randrange(0, len(MT3))])  
 newClass.set\_meetingTime1(dbMgr.get\_meetingTimes()[0])  
  
 def wholefourhours():  
 newClass.set\_meetingTime(MT4[rnd.randrange(0, len(MT4))])  
 newClass.set\_meetingTime1(dbMgr.get\_meetingTimes()[0])  
  
 def splittwohours():  
 newClass.set\_meetingTime(MT1[rnd.randrange(0, len(MT1))])  
 newClass.set\_meetingTime1(MT1v2[rnd.randrange(0, len(MT1v2))])  
  
 def splitthreehours():  
 newClass.set\_meetingTime(MT1p5[rnd.randrange(0, len(MT1p5))])  
 newClass.set\_meetingTime1(MT1p5v2[rnd.randrange(0, len(MT1p5v2))])  
  
 def splitfourhours():  
 newClass.set\_meetingTime(MT2[rnd.randrange(0, len(MT2))])  
 newClass.set\_meetingTime1(MT2v2[rnd.randrange(0, len(MT2v2))])  
  
 def call\_funcs\_randomly(funcs):  
 shuffle(funcs)  
 for func in funcs:  
 func()  
  
 *# with mixed Split Time* if (subjects[j].get\_numbHour() == 1):  
 newClass.set\_meetingTime(MT1[rnd.randrange(0, len(MT1))])  
 newClass.set\_meetingTime1(dbMgr.get\_meetingTimes()[0])  
 elif (subjects[j].get\_numbHour() == 2):  
 call\_funcs\_randomly([wholetwohours, splittwohours])  
 elif (subjects[j].get\_numbHour() == 3):  
 call\_funcs\_randomly([wholethreehours, splitthreehours])  
 elif (subjects[j].get\_numbHour() == 4):  
 call\_funcs\_randomly([wholefourhours, splitfourhours])  
 elif (subjects[j].get\_numbHour() == 6):  
 newClass.set\_meetingTime(MT3[rnd.randrange(0, len(MT3))])  
 newClass.set\_meetingTime1(MT3v2[rnd.randrange(0, len(MT3v2))])  
 else:  
 newClass.set\_meetingTime(  
 dbMgr.get\_meetingTimes()[rnd.randrange(0, len(dbMgr.get\_meetingTimes()))])  
 newClass.set\_meetingTime1(  
 dbMgr.get\_meetingTimes()[rnd.randrange(0, len(dbMgr.get\_meetingTimes()))])  
 self.\_classes.append(newClass) *# Add result as new class* return self  
  
 def calculate\_fitness(self):  
 self.\_conflicts=[]  
 classes=self.get\_classes()  
 instructorNames=list()  
 instructor\_Hours=list()  
 for i in range(0, len(classes)):  
 instructorNames.append(classes[i].get\_instructor().get\_name())  
 subjectHours=list()  
 if (classes[i].get\_meetingTime1().get\_id() != **'NULL'**):  
 if (classes[i].get\_meetingTime().get\_id() == classes[i].get\_meetingTime1().get\_id()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(Conflict(Conflict.ConflictType.SAME\_MTS, conflictBetweenClasses))  
 if DISABLE\_UNEQUAL\_SPLIT == True:  
 if (classes[i].get\_meetingTime().get\_MThour() != classes[  
 i].get\_meetingTime1().get\_MThour()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.UNEQUAL\_SPLIT, conflictBetweenClasses))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 i].get\_meetingTime1().get\_restTime()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.OVERLAP\_MTS, conflictBetweenClasses))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 i].get\_meetingTime().get\_restTime()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.OVERLAP\_MTS, conflictBetweenClasses))  
 if DISABLE\_CASUAL\_SPLITTING == True:  
 *# Disable Same Day Split Time* if (classes[i].get\_meetingTime().get\_day() == classes[i].get\_meetingTime1().get\_day()):  
 if (classes[i].get\_meetingTime().get\_MThour() != classes[  
 i].get\_subject().get\_numbHour()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.CASUAL\_SPLITTING, conflictBetweenClasses))  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[i].get\_meetingTime1().get\_MThour()) !=  
 classes[i].get\_subject().get\_numbHour()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.WRONG\_MTHOUR, conflictBetweenClasses))  
 if ENABLE\_NUMB\_OF\_STUDENTS == True:  
 if (classes[i].get\_room().get\_seatingCapacity() < classes[  
 i].get\_subject().get\_maxNumbOfStudents()):  
 seatingCapacityConflict=list()  
 seatingCapacityConflict.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.NUMB\_OF\_STUDENTS, seatingCapacityConflict))  
 *# Disable to assign Labs on VR* if LAB\_ON\_VR == False:  
 if (classes[i].get\_subject().get\_compatibility() == **'PR'**):  
 if (classes[i].get\_room().get\_type() != **'PR'**):  
 roomCompatibilityConflict=list()  
 roomCompatibilityConflict.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.LAB\_ON\_VR, roomCompatibilityConflict))  
 *# Disable cross department room utility* if DISABLE\_XDEPT\_ROOM\_UTILITY == True:  
 if (classes[i].get\_room().get\_type() != **'VR'**):  
 if (classes[i].get\_subject().get\_dept() != classes[i].get\_room().get\_dept()):  
 roomUtilityConflict=list()  
 roomUtilityConflict.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.XDEPT\_ROOM\_UTILITY, roomUtilityConflict))  
  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 0* if (classes[i].get\_meetingTime().get\_id() in classes[  
 i].get\_instructor().get\_overtime()): *# This identify the schedules inside the overtime* if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_OVERTIME, conflictBetweenClasses))  
 else:  
 *# 0* if (classes[i].get\_meetingTime().get\_id() in classes[i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_instructor().get\_availability()): *# This removes the schedules out side the official time and overtime* conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_OVERTIME, conflictBetweenClasses))  
 *# 1* if (classes[i].get\_meetingTime1().get\_id() in classes[i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime1().get\_id() not in classes[  
 i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_OVERTIME, conflictBetweenClasses))  
 *# Limit Instructor Availibilty* if LIMIT\_INSTRUCTOR\_AVAILABILITY == True:  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_AVAILABILITY,  
 conflictBetweenClasses))  
 else:  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_AVAILABILITY,  
 conflictBetweenClasses))  
 *# 1* if (classes[i].get\_meetingTime1().get\_id() not in classes[  
 i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime1().get\_id() not in classes[  
 i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_AVAILABILITY,  
 conflictBetweenClasses))  
 *# Limit Room Availibilty* if LIMIT\_ROOM\_AVAILABILITY == True:  
 if (classes[i].get\_room().get\_type() != **'VR'**):  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_room().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_AVAILABILITY, conflictBetweenClasses))  
 else:  
 *# 1* if (classes[i].get\_meetingTime1().get\_id() not in classes[  
 i].get\_room().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_AVAILABILITY, conflictBetweenClasses))  
 *# Limit Section Availibilty* if LIMIT\_SECTION\_AVAILABILITY == True:  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_section().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_AVAILABILITY, conflictBetweenClasses))  
 else:  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_section().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_AVAILABILITY, conflictBetweenClasses))  
 *# 1* if (classes[i].get\_meetingTime1().get\_id() not in classes[  
 i].get\_section().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_AVAILABILITY, conflictBetweenClasses))  
  
 for j in range(0, len(classes)):  
 *# Class Optimization* if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day() and  
 classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[  
 j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 else:  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day() and  
 classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[  
 j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 *# 11* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime1().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime1().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[  
 j].get\_meetingTime1().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime1().get\_breakTime()):  
 if ((classes[i].get\_meetingTime1().get\_MThour() + classes[  
 j].get\_meetingTime1().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 *# 10* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime1().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime1().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime1().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime1().get\_MThour() + classes[  
 j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 *# 01* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime1().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[  
 j].get\_meetingTime1().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 *# Instructor Optimization* if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[  
 j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 else:  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[  
 j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# 11* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime1().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime1().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[  
 j].get\_meetingTime1().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime1().get\_breakTime()):  
 if ((classes[i].get\_meetingTime1().get\_MThour() + classes[  
 j].get\_meetingTime1().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# 01* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime1().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime1().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime1().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime1().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[  
 j].get\_meetingTime1().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# 10* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime1().get\_MThour() + classes[  
 j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# Room Optimization* if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day() and  
 classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 else:  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day() and  
 classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 *# 11* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime1().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime1().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[  
 j].get\_meetingTime1().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 *# 01* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime1().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime1().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime1().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 *# 10* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 if (j >= i):  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_meetingTime().get\_id() == classes[j].get\_meetingTime().get\_id()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, roomBookingConflict))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 else:  
 *# 00* if (classes[i].get\_meetingTime().get\_id() == classes[j].get\_meetingTime().get\_id()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, roomBookingConflict))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# 11* if (classes[i].get\_meetingTime1().get\_id() == classes[j].get\_meetingTime1().get\_id()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, roomBookingConflict))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# 01* if (classes[i].get\_meetingTime().get\_id() == classes[j].get\_meetingTime1().get\_id()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, roomBookingConflict))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# 10* if (classes[i].get\_meetingTime1().get\_id() == classes[j].get\_meetingTime().get\_id()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, roomBookingConflict))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# Disable Dual Class Type* if DISABLE\_MIXED\_TYPE == True:  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime().get\_day() == classes[  
 j].get\_meetingTime().get\_day()):  
 if (classes[i].get\_room().get\_type() != classes[  
 j].get\_room().get\_type()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 roomBookingConflict)) *# Room Type Conflict* else:  
 *# 00* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime().get\_day() == classes[  
 j].get\_meetingTime().get\_day()):  
 if (classes[i].get\_room().get\_type() != classes[  
 j].get\_room().get\_type()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 roomBookingConflict)) *# Room Type Conflict  
 # 11* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime1().get\_day() == classes[  
 j].get\_meetingTime1().get\_day()):  
 if (classes[i].get\_room().get\_type() != classes[  
 j].get\_room().get\_type()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 roomBookingConflict)) *# Room Type Conflict  
 # 01* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime().get\_day() == classes[  
 j].get\_meetingTime1().get\_day()):  
 if (classes[i].get\_room().get\_type() != classes[  
 j].get\_room().get\_type()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 roomBookingConflict)) *# Room Type Conflict  
 # 10* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime1().get\_day() == classes[  
 j].get\_meetingTime().get\_day()):  
 if (classes[i].get\_room().get\_type() != classes[  
 j].get\_room().get\_type()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 roomBookingConflict)) *# Room Type Conflict* if (classes[j].get\_instructor().get\_name() == instructorNames[i]):  
 subjectHours.append(float(classes[j].get\_subject().get\_numbHour()))  
 *# Enable Unit Limit* instructor\_Hours.append(subjectHours)  
 if DISABLE\_INSTRUCTOR\_OVERLOAD == True:  
 if (classes[i].get\_instructor().get\_max\_hours() < sum(instructor\_Hours[i])):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_OVERLOAD, instructorBookingConflict))  
 return 1 / ((1.0 \* len(self.\_conflicts) + 1))  
  
 def \_\_str\_\_(self):  
 returnValue=**""** for i in range(0, len(self.\_classes) - 1):  
 returnValue+=str(self.\_classes[i]) + **", "** returnValue+=str(self.\_classes[len(self.\_classes) - 1])  
 return returnValue  
  
 class Population:  
 def \_\_init\_\_(self, size):  
 self.\_size=size  
 self.\_data=dbMgr  
 self.\_schedules=[]  
 for i in range(0, size): self.\_schedules.append(Schedule().initialize())  
  
 def get\_size(self): return self.\_size  
  
 def get\_schedules(self): return self.\_schedules  
  
 class GeneticAlgorithm:  
 def evolve(self, population):  
 return self.\_mutate\_population(self.\_crossover\_population(population))  
  
 def \_crossover\_population(self, pop):  
 crossover\_pop=Population(0)  
 for i in range(NUMB\_OF\_ELITE\_SCHEDULES):  
 crossover\_pop.get\_schedules().append(pop.get\_schedules()[i])  
 i=NUMB\_OF\_ELITE\_SCHEDULES  
 while i < POPULATION\_SIZE:  
 schedule1=self.\_select\_tournament\_population(pop).get\_schedules()[0]  
 schedule2=self.\_select\_tournament\_population(pop).get\_schedules()[0]  
 crossover\_pop.get\_schedules().append(self.\_crossover\_schedule(schedule1, schedule2))  
 i+=1  
 return crossover\_pop  
  
 def \_mutate\_population(self, population):  
 for i in range(NUMB\_OF\_ELITE\_SCHEDULES, POPULATION\_SIZE):  
 self.\_mutate\_schedule(population.get\_schedules()[i])  
 return population  
  
 def \_crossover\_schedule(self, schedule1, schedule2):  
 crossoverSchedule=Schedule().initialize()  
 for i in range(0, len(crossoverSchedule.get\_classes())):  
 if (rnd.random() > 0.5):  
 crossoverSchedule.get\_classes()[i]=schedule1.get\_classes()[i]  
 else:  
 crossoverSchedule.get\_classes()[i]=schedule2.get\_classes()[i]  
 return crossoverSchedule  
  
 def \_mutate\_schedule(self, mutateSchedule):  
 schedule=Schedule().initialize()  
 for i in range(0, len(mutateSchedule.get\_classes())):  
 if (MUTATION\_RATE > rnd.random()): mutateSchedule.get\_classes()[i]=schedule.get\_classes()[i]  
 return mutateSchedule  
  
 def \_select\_tournament\_population(self, pop):  
 tournament\_pop=Population(0)  
 i=0  
 while i < TOURNAMENT\_SELECTION\_SIZE:  
 tournament\_pop.get\_schedules().append(  
 pop.get\_schedules()[rnd.randrange(0, int(POPULATION\_SIZE))])  
 i+=1  
 tournament\_pop.get\_schedules().sort(key = lambda x: x.get\_fitness(), reverse = True)  
 return tournament\_pop  
  
 class Section:  
 def \_\_init\_\_(self, code, subjects, availability):  
 self.\_code=code  
 self.\_subjects=subjects  
 self.\_availability=availability  
  
 def get\_code(self): return self.\_code  
  
 def get\_subjects(self): return self.\_subjects  
  
 def get\_availability(self): return self.\_availability  
  
 class Subject:  
 def \_\_init\_\_(self, code, description, units, numbHour, compatibility, curriculum, maxNumbOfStudents,  
 dept,  
 instructors):  
 self.\_code=code  
 self.\_description=description  
 self.\_units=units  
 self.\_numbHour=numbHour  
 self.\_compatibility=compatibility  
 self.\_curriculum=curriculum  
 self.\_maxNumbOfStudents=maxNumbOfStudents  
 self.\_dept=dept  
 self.\_instructors=instructors  
  
 def get\_code(self): return self.\_code  
  
 def get\_description(self): return self.\_description  
  
 def get\_units(self): return self.\_units  
  
 def get\_numbHour(self): return self.\_numbHour  
  
 def get\_compatibility(self): return self.\_compatibility  
  
 def get\_curriculum(self): return self.\_curriculum  
  
 def get\_maxNumbOfStudents(self): return self.\_maxNumbOfStudents  
  
 def get\_dept(self): return self.\_dept  
  
 def get\_instructors(self): return self.\_instructors  
  
 def \_\_str\_\_(self): return self.\_description  
  
 class Time:  
 def \_\_init\_\_(self, block):  
 self.\_block=block  
  
 def get\_block(self): return self.\_block  
  
 def \_\_str\_\_(self): return self.\_block  
  
 class TimePoint:  
 def \_\_init\_\_(self, point, blocks, phantomBlocks):  
 self.\_point=point  
 self.\_blocks=blocks  
 self.\_phantomBlocks=phantomBlocks  
  
 def get\_point(self): return self.\_point  
  
 def get\_blocks(self): return self.\_blocks  
  
 def get\_phantomBlocks(self): return self.\_phantomBlocks  
  
 def \_\_str\_\_(self): return self.\_point  
  
 class Instructor:  
 def \_\_init\_\_(self, number, name, max\_hours, seniority, availability, start, end, overtime):  
 self.\_number=number  
 self.\_name=name  
 self.\_max\_hours=max\_hours  
 self.\_seniority=seniority  
 self.\_availability=availability  
 self.\_start=start  
 self.\_end=end  
 self.\_overtime=overtime  
  
 def get\_number(self): return self.\_number  
  
 def get\_name(self): return self.\_name  
  
 def get\_max\_hours(self): return self.\_max\_hours  
  
 def get\_seniority(self): return self.\_seniority  
  
 def get\_availability(self): return self.\_availability  
  
 def get\_start(self): return self.\_start  
  
 def get\_end(self): return self.\_end  
  
 def get\_overtime(self): return self.\_overtime  
  
 def \_\_str\_\_(self): return self.\_name  
  
 class MeetingTime:  
 def \_\_init\_\_(self, id, time, day, MThour, cell, restTime, breakTime):  
 self.\_id=id  
 self.\_time=time  
 self.\_day=day  
 self.\_MThour=MThour  
 self.\_cell=cell  
 self.\_restTime=restTime  
 self.\_breakTime=breakTime  
  
 def get\_id(self): return self.\_id  
  
 def get\_time(self): return self.\_time  
  
 def get\_day(self): return self.\_day  
  
 def get\_MThour(self): return self.\_MThour  
  
 def get\_cell(self): return self.\_cell  
  
 def get\_restTime(self): return self.\_restTime  
  
 def get\_breakTime(self): return self.\_breakTime  
  
 def \_\_str\_\_(self): return self.\_id  
  
 class MeetingTime1:  
 def \_\_init\_\_(self, id, time, day, MThour, cell, restTime, breakTime):  
 self.\_id=id  
 self.\_time=time  
 self.\_day=day  
 self.\_MThour=MThour  
 self.\_cell=cell  
 self.\_restTime=restTime  
 self.\_breakTime=breakTime  
  
 def get\_id(self): return self.\_id  
  
 def get\_time(self): return self.\_time  
  
 def get\_day(self): return self.\_day  
  
 def get\_MThour(self): return self.\_MThour  
  
 def get\_cell(self): return self.\_cell  
  
 def get\_restTime(self): return self.\_restTime  
  
 def get\_breakTime(self): return self.\_breakTime  
  
 def \_\_str\_\_(self): return self.\_id  
  
 class Room:  
 def \_\_init\_\_(self, number, seatingCapacity, type, dept, availability):  
 self.\_number=number  
 self.\_seatingCapacity=seatingCapacity  
 self.\_type=type  
 self.\_dept=dept  
 self.\_availability=availability  
  
 def get\_number(self): return self.\_number  
  
 def get\_seatingCapacity(self): return self.\_seatingCapacity  
  
 def get\_type(self): return self.\_type  
  
 def get\_dept(self): return self.\_dept  
  
 def get\_availability(self): return self.\_availability  
  
 def \_\_str\_\_(self): return self.\_number  
  
 class Class:  
 def \_\_init\_\_(self, id, section, subject):  
 self.\_id=id  
 self.\_section=section  
 self.\_subject=subject  
 self.\_instructor=None  
 self.\_meetingTime=None  
 self.\_meetingTime1=None  
 self.\_room=None  
  
 def get\_id(self): return self.\_id  
  
 def get\_section(self): return self.\_section  
  
 def get\_subject(self): return self.\_subject  
  
 def get\_instructor(self): return self.\_instructor  
  
 def get\_meetingTime(self): return self.\_meetingTime  
  
 def get\_meetingTime1(self): return self.\_meetingTime1  
  
 def get\_room(self): return self.\_room  
  
 def set\_instructor(self, instructor): self.\_instructor=instructor  
  
 def set\_meetingTime(self, meetingTime): self.\_meetingTime=meetingTime  
  
 def set\_meetingTime1(self, meetingTime1): self.\_meetingTime1=meetingTime1  
  
 def set\_room(self, room): self.\_room=room  
  
 def \_\_str\_\_(self):  
 return str(self.\_section.get\_code()) + **","** + str(self.\_subject.get\_code()) + **","** + \  
 str(self.\_room.get\_number()) + **","** + str(self.\_instructor.get\_number()) + **","** + str(  
 self.\_meetingTime.get\_id()) + **","** + str(self.\_meetingTime1.get\_id())  
  
 class Conflict:  
 class ConflictType(Enum):  
 WRONG\_MTHOUR=1  
 UNEQUAL\_SPLIT=2  
 CASUAL\_SPLITTING=3  
 SAME\_MTS=4  
 OVERLAP\_MTS=5  
  
 NUMB\_OF\_STUDENTS=6  
 ROOM\_AVAILABILITY=7  
 ROOM\_BOOKING=8  
 XDEPT\_ROOM\_UTILITY=9  
  
 INSTRUCTOR\_OVERTIME=10  
 INSTRUCTOR\_AVAILABILITY=11  
 INSTRUCTOR\_BOOKING=12  
 INSTRUCTOR\_OVERLOAD=13  
  
 SECTION\_AVAILABILITY=14  
 SECTION\_BOOKING=15  
  
 LAB\_ON\_VR=16  
 MIXED\_TYPE=17  
  
 def \_\_init\_\_(self, conflictType, conflictBetweenClasses):  
 self.\_conflictType=conflictType  
 self.\_conflictBetweenClasses=conflictBetweenClasses  
  
 def get\_conflictType(self): return self.\_conflictType  
  
 def get\_conflictBetweenClasses(self): return self.\_conflictBetweenClasses  
  
 def \_\_str\_\_(self): return str(self.\_conflictType) + **" "** + str(  
 **" and "**.join(map(str, self.\_conflictBetweenClasses)))  
  
 def click(event):  
 try:  
 selected=tree.focus() *# Grab record position/number* values=tree.item(selected, **'values'**) *# Grab record values* cb\_section\_code.delete(0, **'end'**)  
 cb\_meeting\_time.delete(0, **'end'**)  
 cb\_section\_code.insert(0, values[0])  
 cb\_meeting\_time.insert(0, values[1])  
 except IndexError: pass  
  
 def combobox\_input0():  
 ARADB()  
 cursor.execute(**"SELECT code FROM `section`"**)  
 conn.commit()  
 data=[]  
 for row in cursor.fetchall():  
 data.append(row[0])  
 return data  
 cursor.close()  
 conn.close()  
  
 def combobox\_input1():  
 ARADB()  
 cursor.execute(**"SELECT id FROM `meeting\_time`"**)  
 conn.commit()  
 data=[]  
 for row in cursor.fetchall():  
 data.append(row[0])  
 return data  
 cursor.close()  
 conn.close()  
  
 def combobox\_input2():  
 ARADB()  
 cursor.execute(**"SELECT point FROM `time\_point`"**)  
 conn.commit()  
 data=[]  
 for row in cursor.fetchall():  
 data.append(row[0])  
 return data  
 cursor.close()  
 conn.close()  
  
 def Create(): *# MT INPUTS* if SECTION\_CODE.get() == **""**:  
 txt\_result.config(text = **"Please choose a section!"**, fg = **"red"**)  
 else:  
 dbMgr=DBMgr()  
 meetingTimes=dbMgr.get\_meetingTimes()  
 timepoints=dbMgr.get\_timepoints()  
 sep=**' '** if (not (MON\_START.get() == **""**)):  
 ID\_Mon=list() *# Meeting Time IDs in Monday* Time\_Mon=list()  
 startingTime\_Mon=list()  
 endingTime\_Mon=list()  
 phantom\_time=list()  
 qualifiedMTs\_Mon=list()  
 qualifiedBlocks\_Mon=list()  
 *# List all Mondays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Monday'** == meetingTimes[i].get\_day()):  
 ID\_Mon.append(meetingTimes[i].get\_id())  
 Time\_Mon.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (MON\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Mon.append(timepoints[i].get\_blocks())  
 if (MON\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Mon.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (MON\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Mon)):  
 if (Time\_Mon[i] in startingTime\_Mon[0]):  
 if (Time\_Mon[i] not in endingTime\_Mon[0]):  
 if (Time\_Mon[i] not in phantom\_time[0]):  
 qualifiedMTs\_Mon.append(ID\_Mon[i])  
 qualifiedBlocks\_Mon.append(Time\_Mon[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Mon)):  
 if (Time\_Mon[i] in qualifiedBlocks\_Mon):  
 cursor.execute(  
 **"INSERT INTO 'section\_availability' (section\_code, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(SECTION\_CODE.get().split(sep, 1)[0]), str(ID\_Mon[i]), str(MON\_START.get()),  
 str(MON\_END.get())))  
 conn.commit()  
 MON\_START.set(**""**)  
 MON\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 if (not (TUE\_START.get() == **""**)):  
 ID\_Tue=list() *# Meeting Time IDs in Tuesday* Time\_Tue=list()  
 startingTime\_Tue=list()  
 endingTime\_Tue=list()  
 phantom\_time=list()  
 qualifiedMTs\_Tue=list()  
 qualifiedBlocks\_Tue=list()  
 *# List all Tuesdays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Tuesday'** == meetingTimes[i].get\_day()):  
 ID\_Tue.append(meetingTimes[i].get\_id())  
 Time\_Tue.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (TUE\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Tue.append(timepoints[i].get\_blocks())  
 if (TUE\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Tue.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (TUE\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Tue)):  
 if (Time\_Tue[i] in startingTime\_Tue[0]):  
 if (Time\_Tue[i] not in endingTime\_Tue[0]):  
 if (Time\_Tue[i] not in phantom\_time[0]):  
 qualifiedMTs\_Tue.append(ID\_Tue[i])  
 qualifiedBlocks\_Tue.append(Time\_Tue[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Tue)):  
 if (Time\_Tue[i] in qualifiedBlocks\_Tue):  
 cursor.execute(  
 **"INSERT INTO 'section\_availability' (section\_code, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(SECTION\_CODE.get().split(sep, 1)[0]), str(ID\_Tue[i]), str(TUE\_START.get()),  
 str(TUE\_END.get())))  
 conn.commit()  
 TUE\_START.set(**""**)  
 TUE\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 if (not (WED\_START.get() == **""**)):  
 ID\_Wed=list() *# Meeting Time IDs in Wednesday* Time\_Wed=list()  
 startingTime\_Wed=list()  
 endingTime\_Wed=list()  
 phantom\_time=list()  
 qualifiedMTs\_Wed=list()  
 qualifiedBlocks\_Wed=list()  
 *# List all Wednesdays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Wednesday'** == meetingTimes[i].get\_day()):  
 ID\_Wed.append(meetingTimes[i].get\_id())  
 Time\_Wed.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (WED\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Wed.append(timepoints[i].get\_blocks())  
 if (WED\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Wed.append(timepoints[i].get\_blocks())  
 **"""print("Number of Added Blocks: " + str(len(startingTime\_Wed[0])))  
 print("Number of Deleted Blocks: " + str(len(endingTime\_Wed[0])))"""** *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (WED\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Wed)):  
 if (Time\_Wed[i] in startingTime\_Wed[0]):  
 if (Time\_Wed[i] not in endingTime\_Wed[0]):  
 if (Time\_Wed[i] not in phantom\_time[0]):  
 qualifiedMTs\_Wed.append(ID\_Wed[i])  
 qualifiedBlocks\_Wed.append(Time\_Wed[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Wed)):  
 if (Time\_Wed[i] in qualifiedBlocks\_Wed):  
 cursor.execute(  
 **"INSERT INTO 'section\_availability' (section\_code, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(SECTION\_CODE.get().split(sep, 1)[0]), str(ID\_Wed[i]), str(WED\_START.get()),  
 str(WED\_END.get())))  
 conn.commit()  
 WED\_START.set(**""**)  
 WED\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 if (not (THU\_START.get() == **""**)):  
 ID\_Thu=list() *# Meeting Time IDs in Thursday* Time\_Thu=list()  
 startingTime\_Thu=list()  
 endingTime\_Thu=list()  
 phantom\_time=list()  
 qualifiedMTs\_Thu=list()  
 qualifiedBlocks\_Thu=list()  
 *# List all Thursdays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Thursday'** == meetingTimes[i].get\_day()):  
 ID\_Thu.append(meetingTimes[i].get\_id())  
 Time\_Thu.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (THU\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Thu.append(timepoints[i].get\_blocks())  
 if (THU\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Thu.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (THU\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Thu)):  
 if (Time\_Thu[i] in startingTime\_Thu[0]):  
 if (Time\_Thu[i] not in endingTime\_Thu[0]):  
 if (Time\_Thu[i] not in phantom\_time[0]):  
 qualifiedMTs\_Thu.append(ID\_Thu[i])  
 qualifiedBlocks\_Thu.append(Time\_Thu[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Thu)):  
 if (Time\_Thu[i] in qualifiedBlocks\_Thu):  
 cursor.execute(  
 **"INSERT INTO 'section\_availability' (section\_code, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(SECTION\_CODE.get().split(sep, 1)[0]), str(ID\_Thu[i]), str(THU\_START.get()),  
 str(THU\_END.get())))  
 conn.commit()  
 THU\_START.set(**""**)  
 THU\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 if (not (FRI\_START.get() == **""**)):  
 ID\_Fri=list() *# Meeting Time IDs in Friday* Time\_Fri=list()  
 startingTime\_Fri=list()  
 endingTime\_Fri=list()  
 phantom\_time=list()  
 qualifiedMTs\_Fri=list()  
 qualifiedBlocks\_Fri=list()  
 *# List all Fridays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Friday'** == meetingTimes[i].get\_day()):  
 ID\_Fri.append(meetingTimes[i].get\_id())  
 Time\_Fri.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (FRI\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Fri.append(timepoints[i].get\_blocks())  
 if (FRI\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Fri.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (FRI\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Fri)):  
 if (Time\_Fri[i] in startingTime\_Fri[0]):  
 if (Time\_Fri[i] not in endingTime\_Fri[0]):  
 if (Time\_Fri[i] not in phantom\_time[0]):  
 qualifiedMTs\_Fri.append(ID\_Fri[i])  
 qualifiedBlocks\_Fri.append(Time\_Fri[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Fri)):  
 if (Time\_Fri[i] in qualifiedBlocks\_Fri):  
 cursor.execute(  
 **"INSERT INTO 'section\_availability' (section\_code, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(SECTION\_CODE.get().split(sep, 1)[0]), str(ID\_Fri[i]), str(FRI\_START.get()),  
 str(FRI\_END.get())))  
 conn.commit()  
 FRI\_START.set(**""**)  
 FRI\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 if (not (SAT\_START.get() == **""**)):  
 ID\_Sat=list() *# Meeting Time IDs in Saturday* Time\_Sat=list()  
 startingTime\_Sat=list()  
 endingTime\_Sat=list()  
 phantom\_time=list()  
 qualifiedMTs\_Sat=list()  
 qualifiedBlocks\_Sat=list()  
 *# List all Saturdays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Saturday'** == meetingTimes[i].get\_day()):  
 ID\_Sat.append(meetingTimes[i].get\_id())  
 Time\_Sat.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (SAT\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Sat.append(timepoints[i].get\_blocks())  
 if (SAT\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Sat.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (SAT\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Sat)):  
 if (Time\_Sat[i] in startingTime\_Sat[0]):  
 if (Time\_Sat[i] not in endingTime\_Sat[0]):  
 if (Time\_Sat[i] not in phantom\_time[0]):  
 qualifiedMTs\_Sat.append(ID\_Sat[i])  
 qualifiedBlocks\_Sat.append(Time\_Sat[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Sat)):  
 if (Time\_Sat[i] in qualifiedBlocks\_Sat):  
 cursor.execute(  
 **"INSERT INTO 'section\_availability' (section\_code, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(SECTION\_CODE.get().split(sep, 1)[0]), str(ID\_Sat[i]), str(SAT\_START.get()),  
 str(SAT\_END.get())))  
 conn.commit()  
 SAT\_START.set(**""**)  
 SAT\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 if (not (SUN\_START.get() == **""**)):  
 ID\_Sun=list() *# Meeting Time IDs in Sunday* Time\_Sun=list()  
 startingTime\_Sun=list()  
 endingTime\_Sun=list()  
 phantom\_time=list()  
 qualifiedMTs\_Sun=list()  
 qualifiedBlocks\_Sun=list()  
 *# List all Sundays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Sunday'** == meetingTimes[i].get\_day()):  
 ID\_Sun.append(meetingTimes[i].get\_id())  
 Time\_Sun.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (SUN\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Sun.append(timepoints[i].get\_blocks())  
 if (SUN\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Sun.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (SUN\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Sun)):  
 if (Time\_Sun[i] in startingTime\_Sun[0]):  
 if (Time\_Sun[i] not in endingTime\_Sun[0]):  
 if (Time\_Sun[i] not in phantom\_time[0]):  
 qualifiedMTs\_Sun.append(ID\_Sun[i])  
 qualifiedBlocks\_Sun.append(Time\_Sun[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Sun)):  
 if (Time\_Sun[i] in qualifiedBlocks\_Sun):  
 cursor.execute(  
 **"INSERT INTO 'section\_availability' (section\_code, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(SECTION\_CODE.get().split(sep, 1)[0]), str(ID\_Sun[i]), str(SUN\_START.get()),  
 str(SUN\_END.get())))  
 conn.commit()  
 SUN\_START.set(**""**)  
 SUN\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Created a data!"**, fg = **"green"**)  
  
 def View():  
 tree.delete(\*tree.get\_children())  
 ARADB()  
 cursor.execute(**"SELECT \* FROM `section\_availability` ORDER BY `section\_code` ASC"**)  
 fetch=cursor.fetchall()  
 for data in fetch:  
 tree.insert(**''**, **'end'**, values = (data[0], data[1], data[2], data[3]))  
 cursor.close()  
 conn.close()  
 txt\_result.config(text = **"Successfully viewed the data from database"**, fg = **"black"**)  
  
 def Delete():  
 if SECTION\_CODE.get() == **""**:  
 txt\_result.config(text = **"Please choose a section!"**, fg = **"red"**)  
 else:  
 sep=**' '** ARADB()  
 cursor.execute(  
 **"DELETE FROM 'section\_availability' WHERE section\_code = '"** + str(  
 SECTION\_CODE.get().split(sep, 1)[0]) + **"'"**)  
 conn.commit()  
 cb\_section\_code.delete(0, **'end'**)  
 cb\_meeting\_time.delete(0, **'end'**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Deleted Successfully!"**, fg = **"green"**)  
  
 def Clear():  
 result=messagebox.askquestion(**"Clear"**, **"This will clear the Section Availability table.**\n**Do you want to proceed?"**, icon = **'warning'**)  
 if result == **'yes'**:  
 ARADB()  
 cursor.execute(**"""DROP TABLE section\_availability"""**)  
 cursor.execute(**"""create table section\_availability (section\_code text NOT NULL, meeting\_time\_id text NOT NULL, start text, end text)"""**)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Table Cleared Successfully!"**, fg = **"green"**)  
 else:  
 pass  
  
 def Refresh():  
 SA.destroy()  
 Modify\_Availability()  
  
 *# ==================================VARIABLES==========================================* SECTION\_CODE=StringVar(SA)  
 MEETING\_TIME=StringVar(SA)  
 MON\_START=StringVar(SA)  
 MON\_END=StringVar(SA)  
 TUE\_START=StringVar(SA)  
 TUE\_END=StringVar(SA)  
 WED\_START=StringVar(SA)  
 WED\_END=StringVar(SA)  
 THU\_START=StringVar(SA)  
 THU\_END=StringVar(SA)  
 FRI\_START=StringVar(SA)  
 FRI\_END=StringVar(SA)  
 SAT\_START=StringVar(SA)  
 SAT\_END=StringVar(SA)  
 SUN\_START=StringVar(SA)  
 SUN\_END=StringVar(SA)  
  
 *# ==================================FRAME==============================================* Top=Frame(SA, width = 900, height = 50, bd = 8, relief = **"raise"**, bg = **"#CC99C9"**)  
 Top.pack(side = TOP)  
 Left=Frame(SA, width = 300, height = 500, bd = 8, relief = **"raise"**)  
 Left.pack(side = LEFT)  
 Right=Frame(SA, width = 600, height = 500, bd = 8, relief = **"raise"**)  
 Right.pack(side = RIGHT)  
 Forms=Frame(Left, width = 300, height = 450)  
 Forms.pack(side = TOP)  
 Buttons=Frame(Left, width = 300, height = 100, bd = 8, relief = **"raise"**)  
 Buttons.pack(side = BOTTOM)  
  
 *# ==================================LABEL WIDGET=======================================* txt\_title=Label(Top, width = 900, font = (**'arial'**, 24), text = **"Section Availability"**, fg = **"Black"**,  
 bg = **"#8e9ae6"**)  
 txt\_title.pack()  
 txt\_section=Label(Forms, text = **"Select Section:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_section.grid(row = 0, stick = **"e"**)  
  
 txt\_monday=Label(Forms, text = **"Monday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_monday.grid(row = 1, stick = **"e"**)  
 txt\_tuesday=Label(Forms, text = **"Tuesday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_tuesday.grid(row = 2, stick = **"e"**)  
 txt\_wednesday=Label(Forms, text = **"Wednesday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_wednesday.grid(row = 3, stick = **"e"**)  
 txt\_thursday=Label(Forms, text = **"Thursday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_thursday.grid(row = 4, stick = **"e"**)  
 txt\_friday=Label(Forms, text = **"Friday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_friday.grid(row = 5, stick = **"e"**)  
 txt\_saturday=Label(Forms, text = **"Saturday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_saturday.grid(row = 6, stick = **"e"**)  
 txt\_sunday=Label(Forms, text = **"Sunday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_sunday.grid(row = 7, stick = **"e"**)  
 txt\_result=Label(Buttons)  
 txt\_result.pack(side = TOP)  
  
 *# ==================================ENTRY WIDGET=======================================* cb\_section\_code=ttk.Combobox(Forms, textvariable = SECTION\_CODE, width = 44)  
 cb\_section\_code[**'values'**]=combobox\_input0()  
 cb\_section\_code.place(x = 194, y = 18)  
 cb\_meeting\_time=ttk.Combobox(Forms, textvariable = MEETING\_TIME, width = 20)  
 cb\_meeting\_time[**'values'**]=combobox\_input1()  
  
 txt\_monstart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_monstart.grid(column = 1, row = 1, stick = **"w"**)  
 cb\_monstart=ttk.Combobox(Forms, textvariable = MON\_START, width = 10)  
 cb\_monstart[**'values'**]=combobox\_input2()  
 cb\_monstart.grid(column = 2, row = 1, stick = **"w"**)  
 txt\_monend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 12)  
 txt\_monend.grid(column = 3, row = 1, stick = **"w"**)  
 cb\_monend=ttk.Combobox(Forms, textvariable = MON\_END, width = 10)  
 cb\_monend[**'values'**]=combobox\_input2()  
 cb\_monend.grid(column = 4, row = 1, padx = 10)  
  
 txt\_tuestart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_tuestart.grid(column = 1, row = 2, stick = **"w"**)  
 cb\_tuestart=ttk.Combobox(Forms, textvariable = TUE\_START, width = 10)  
 cb\_tuestart[**'values'**]=combobox\_input2()  
 cb\_tuestart.grid(column = 2, row = 2, stick = **"w"**)  
 txt\_tueend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_tueend.grid(column = 3, row = 2, stick = **"w"**)  
 cb\_tueend=ttk.Combobox(Forms, textvariable = TUE\_END, width = 10)  
 cb\_tueend[**'values'**]=combobox\_input2()  
 cb\_tueend.grid(column = 4, row = 2, padx = 10)  
  
 txt\_wedstart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_wedstart.grid(column = 1, row = 3, stick = **"w"**)  
 cb\_wedstart=ttk.Combobox(Forms, textvariable = WED\_START, width = 10)  
 cb\_wedstart[**'values'**]=combobox\_input2()  
 cb\_wedstart.grid(column = 2, row = 3, stick = **"w"**)  
 txt\_wedend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_wedend.grid(column = 3, row = 3, stick = **"w"**)  
 cb\_wedend=ttk.Combobox(Forms, textvariable = WED\_END, width = 10)  
 cb\_wedend[**'values'**]=combobox\_input2()  
 cb\_wedend.grid(column = 4, row = 3, padx = 10)  
  
 txt\_thustart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_thustart.grid(column = 1, row = 4, stick = **"w"**)  
 cb\_thustart=ttk.Combobox(Forms, textvariable = THU\_START, width = 10)  
 cb\_thustart[**'values'**]=combobox\_input2()  
 cb\_thustart.grid(column = 2, row = 4, stick = **"w"**)  
 txt\_thuend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_thuend.grid(column = 3, row = 4, stick = **"w"**)  
 cb\_thuend=ttk.Combobox(Forms, textvariable = THU\_END, width = 10)  
 cb\_thuend[**'values'**]=combobox\_input2()  
 cb\_thuend.grid(column = 4, row = 4, padx = 10)  
  
 txt\_fristart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_fristart.grid(column = 1, row = 5, stick = **"w"**)  
 cb\_fristart=ttk.Combobox(Forms, textvariable = FRI\_START, width = 10)  
 cb\_fristart[**'values'**]=combobox\_input2()  
 cb\_fristart.grid(column = 2, row = 5, stick = **"w"**)  
 txt\_friend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_friend.grid(column = 3, row = 5, stick = **"w"**)  
 cb\_friend=ttk.Combobox(Forms, textvariable = FRI\_END, width = 10)  
 cb\_friend[**'values'**]=combobox\_input2()  
 cb\_friend.grid(column = 4, row = 5, padx = 10)  
  
 txt\_satstart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_satstart.grid(column = 1, row = 6, stick = **"w"**)  
 cb\_satstart=ttk.Combobox(Forms, textvariable = SAT\_START, width = 10)  
 cb\_satstart[**'values'**]=combobox\_input2()  
 cb\_satstart.grid(column = 2, row = 6, stick = **"w"**)  
 txt\_satend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_satend.grid(column = 3, row = 6, stick = **"w"**)  
 cb\_satend=ttk.Combobox(Forms, textvariable = SAT\_END, width = 10)  
 cb\_satend[**'values'**]=combobox\_input2()  
 cb\_satend.grid(column = 4, row = 6, padx = 10)  
  
 txt\_sunstart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_sunstart.grid(column = 1, row = 7, stick = **"w"**)  
 cb\_sunstart=ttk.Combobox(Forms, textvariable = SUN\_START, width = 10)  
 cb\_sunstart[**'values'**]=combobox\_input2()  
 cb\_sunstart.grid(column = 2, row = 7, stick = **"w"**)  
 txt\_sunend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_sunend.grid(column = 3, row = 7, stick = **"w"**)  
 cb\_sunend=ttk.Combobox(Forms, textvariable = SUN\_END, width = 10)  
 cb\_sunend[**'values'**]=combobox\_input2()  
 cb\_sunend.grid(column = 4, row = 7, padx = 10)  
  
 *# ==================================BUTTONS WIDGET=====================================* btn\_create=Button(Buttons, width = 10, text = **"Create"**, command = Create)  
 btn\_create.pack(side = LEFT)  
 btn\_view=Button(Buttons, width = 10, text = **"View"**, command = View)  
 btn\_view.pack(side = LEFT)  
 btn\_update=Button(Buttons, width = 10, text = **"Update"**, state = DISABLED)  
 btn\_update.pack(side = LEFT)  
 btn\_delete=Button(Buttons, width = 10, text = **"Delete"**, command = Delete)  
 btn\_delete.pack(side = LEFT)  
 btn\_clear=Button(Buttons, width = 10, text = **"Clear"**, fg = **"Black"**, bg = **"#F2B6AE"**, command = Clear)  
 btn\_clear.pack(side = LEFT)  
 btn\_refresh=Button(Top, width = 10, text = **"Refresh (F5)"**, font = (**'arial'**, 10, **'bold'**), fg = **"White"**,  
 bg = **"green"**,  
 command = Refresh)  
 btn\_refresh.pack(side = RIGHT)  
  
 *# ==================================LIST WIDGET========================================* scrollbary=Scrollbar(Right, orient = VERTICAL)  
 scrollbarx=Scrollbar(Right, orient = HORIZONTAL)  
 columns=(**"Section"**, **"Meeting Time"**, **"Start"**, **"End"**)  
 tree=ttk.Treeview(Right, columns = columns, selectmode = **"extended"**, height = 500,  
 yscrollcommand = scrollbary.set, xscrollcommand = scrollbarx.set)  
  
 def treeview\_sort\_column(tree, col, reverse):  
 l=[(tree.set(k, col), k) for k in tree.get\_children(**''**)]  
 l.sort(reverse = reverse)  
  
 *# rearrange items in sorted positions* for index, (val, k) in enumerate(l):  
 tree.move(k, **''**, index)  
  
 *# reverse sort next time* tree.heading(col, command = lambda \_col=col: treeview\_sort\_column(tree, \_col, not reverse))  
  
 for col in columns:  
 tree.heading(col, text = col, command = lambda \_col=col: \  
 treeview\_sort\_column(tree, \_col, False))  
 *# for TREE Scroll Bars* scrollbary.config(command = tree.yview)  
 scrollbary.pack(side = RIGHT, fill = Y)  
 scrollbarx.config(command = tree.xview)  
 scrollbarx.pack(side = BOTTOM, fill = X)  
 *# inside the tree view* tree.heading(**'Section'**, text = **"Section"**, anchor = W)  
 tree.heading(**'Meeting Time'**, text = **"Meeting Time"**, anchor = W)  
 tree.heading(**'Start'**, text = **"Start"**, anchor = W)  
 tree.heading(**'End'**, text = **"End"**, anchor = W)  
 tree.column(**'#0'**, stretch = NO, minwidth = 0, width = 0)  
 tree.column(**'#1'**, stretch = NO, minwidth = 75, width = 75)  
 tree.column(**'#2'**, stretch = NO, minwidth = 160, width = 160)  
 tree.column(**'#3'**, stretch = NO, minwidth = 60, width = 60)  
 tree.bind(**'<ButtonRelease-1>'**, click)  
 tree.pack()  
  
 if \_\_name\_\_ == **'\_\_main\_\_'**:  
 View()  
 SA.mainloop()  
  
 def Department():  
 DP = Toplevel(root)  
 DP.title(**"ECE-SPV"**)  
 DP.wm\_iconbitmap(**"ece-spv.ico"**)  
 DP.screen\_width = DP.winfo\_screenwidth()  
 DP.screen\_height = DP.winfo\_screenheight()  
 DP.width = 720  
 DP.height = 627  
 DP.x = (screen\_width / 2) - (width / 2)  
 DP.y = (screen\_height / 2) - (height / 2)  
 DP.geometry(**'%dx%d+%d+%d'** % (width, height, x, y))  
 DP.resizable(0, 0)  
  
 *# ==================================COMMANDS============================================* def click(event):  
 try:  
 selected = tree.focus() *# Grab record position/number* values = tree.item(selected, **'values'**) *# Grab record values* cb\_code.delete(0, **'end'**)  
 cb\_code.insert(0, values[0])  
 except IndexError:  
 pass  
  
 def combobox\_input1():  
 ARADB()  
 cursor.execute(**"SELECT code FROM `dept`"**) *# SELECT code FROM `physical\_room`* conn.commit()  
 data = []  
 for row in cursor.fetchall():  
 data.append(row[0])  
 return data  
 cursor.close()  
 conn.close()  
  
 def Create():  
 if CODE.get() == **""**:  
 txt\_result.config(text = **"Please enter a Department Code!"**, fg = **"red"**)  
 else:  
 ARADB()  
 cursor.execute(**"INSERT INTO `dept` (code) VALUES(?)"**, (str(CODE.get()),))  
 conn.commit()  
 CODE.set(**""**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Created a data!"**, fg = **"green"**)  
  
 def View():  
 tree.delete(\*tree.get\_children())  
 ARADB()  
 cursor.execute(**"SELECT \* FROM `dept` ORDER BY `code`"**)  
 fetch = cursor.fetchall()  
 for data in fetch:  
 tree.insert(**''**, **'end'**, values = (data[0]))  
 cursor.close()  
 conn.close()  
 txt\_result.config(text = **"Successfully viewed the data from database"**, fg = **"black"**)  
  
 def Delete():  
 if CODE.get() == **""**:  
 txt\_result.config(text = **"Please enter a Department Code!"**, fg = **"red"**)  
 else:  
 ARADB()  
 cursor.execute(**"DELETE FROM 'dept' WHERE code = '"** + str(CODE.get()) + **"'"**)  
 conn.commit()  
 cb\_code.delete(0, **'end'**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Deleted Successfully!"**, fg = **"green"**)  
  
 def Clear():  
 result = messagebox.askquestion(**"Clear"**, **"This will clear the Section table.**\n**Do you want to proceed?"**,  
 icon = **'warning'**)  
 if result == **'yes'**:  
 ARADB()  
 cursor.execute(**"""DROP TABLE dept"""**)  
 cursor.execute(**"""create table dept (code text PRIMARY KEY)"""**)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Table Cleared Successfully!"**, fg = **"green"**)  
 else:  
 pass  
  
 def Refresh():  
 DP.destroy()  
 Department()  
  
 *# ==================================VARIABLES==========================================* CODE = StringVar(DP)  
  
 *# ==================================FRAME==============================================* Top = Frame(DP, width = 900, height = 50, bd = 8, relief = **"raise"**, bg = **"#CC99C9"**)  
 Top.pack(side = TOP)  
 Left = Frame(DP, width = 300, height = 500, bd = 8, relief = **"raise"**)  
 Left.pack(side = LEFT)  
 Right = Frame(DP, width = 600, height = 500, bd = 8, relief = **"raise"**)  
 Right.pack(side = RIGHT)  
 Forms = Frame(Left, width = 300, height = 450)  
 Forms.pack(side = TOP)  
 Buttons = Frame(Left, width = 300, height = 100, bd = 8, relief = **"raise"**)  
 Buttons.pack(side = BOTTOM)  
  
 *# ==================================LABEL WIDGET=======================================* txt\_title = Label(Top, width = 900, font = (**'arial'**, 24), text = **"Department"**, fg = **"Black"**,  
 bg = **"#a18ee6"**)  
 txt\_title.pack()  
 txt\_code = Label(Forms, text = **"Department:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_code.grid(row = 0, stick = **"e"**)  
 txt\_result = Label(Buttons)  
 txt\_result.pack(side = TOP)  
  
 *# ==================================ENTRY WIDGET=======================================* cb\_code = ttk.Combobox(Forms, textvariable = CODE, width = 20)  
 cb\_code[**'values'**] = combobox\_input1()  
 cb\_code.grid(column = 1, row = 0)  
  
 *# ==================================BUTTONS WIDGET=====================================* btn\_create = Button(Buttons, width = 10, text = **"Create"**, command = Create)  
 btn\_create.pack(side = LEFT)  
 btn\_view = Button(Buttons, width = 10, text = **"View"**, command = View)  
 btn\_view.pack(side = LEFT)  
 btn\_update = Button(Buttons, width = 10, text = **"Update"**, state = DISABLED)  
 btn\_update.pack(side = LEFT)  
 btn\_delete = Button(Buttons, width = 10, text = **"Delete"**, command = Delete)  
 btn\_delete.pack(side = LEFT)  
 btn\_clear = Button(Buttons, width = 10, text = **"Clear"**, fg = **"Black"**, bg = **"#F2B6AE"**, command = Clear)  
 btn\_clear.pack(side = LEFT)  
 btn\_refresh = Button(Top, width = 10, text = **"Refresh (F5)"**, font = (**'arial'**, 10, **'bold'**), fg = **"White"**,  
 bg = **"green"**, command = Refresh)  
 btn\_refresh.pack(side = RIGHT)  
  
 *# ==================================LIST WIDGET========================================* scrollbary = Scrollbar(Right, orient = VERTICAL)  
 scrollbarx = Scrollbar(Right, orient = HORIZONTAL)  
 columns = (**"Department"**,)  
 tree = ttk.Treeview(Right, columns = columns, selectmode = **"extended"**, height = 500,  
 yscrollcommand = scrollbary.set, xscrollcommand = scrollbarx.set)  
  
 *# Column Sorter* def treeview\_sort\_column(tree, col, reverse):  
 l = [(tree.set(k, col), k) for k in tree.get\_children(**''**)]  
 l.sort(reverse = reverse)  
  
 *# rearrange items in sorted positions* for index, (val, k) in enumerate(l):  
 tree.move(k, **''**, index)  
  
 *# reverse sort next time* tree.heading(col, command = lambda \_col=col: treeview\_sort\_column(tree, \_col, not reverse))  
  
 for col in columns:  
 tree.heading(col, text = col, command = lambda \_col=col: \  
 treeview\_sort\_column(tree, \_col, False))  
 *# for TREE Scroll Bars* scrollbary.config(command = tree.yview)  
 scrollbary.pack(side = RIGHT, fill = Y)  
 scrollbarx.config(command = tree.xview)  
 scrollbarx.pack(side = BOTTOM, fill = X)  
 *# inside the tree view* tree.heading(**"Department"**, text = **"Department"**, anchor = W)  
 tree.column(**'#0'**, stretch = NO, minwidth = 0, width = 0)  
 tree.bind(**'<ButtonRelease-1>'**, click)  
 tree.pack()  
  
 if \_\_name\_\_ == **'\_\_main\_\_'**:  
 View()  
 DP.mainloop()  
  
 def Refresh():  
 SC.destroy()  
 Add\_Section()  
  
 *# ==================================VARIABLES==========================================* CODE=StringVar(SC)  
  
 *# ==================================FRAME==============================================* Top=Frame(SC, width = 900, height = 50, bd = 8, relief = **"raise"**, bg = **"#CC99C9"**)  
 Top.pack(side = TOP)  
 Left=Frame(SC, width = 300, height = 500, bd = 8, relief = **"raise"**)  
 Left.pack(side = LEFT)  
 Right=Frame(SC, width = 600, height = 500, bd = 8, relief = **"raise"**)  
 Right.pack(side = RIGHT)  
 Forms=Frame(Left, width = 300, height = 450)  
 Forms.pack(side = TOP)  
 Buttons=Frame(Left, width = 300, height = 100, bd = 8, relief = **"raise"**)  
 Buttons.pack(side = BOTTOM)  
  
 *# ==================================LABEL WIDGET=======================================* txt\_title=Label(Top, width = 900, font = (**'arial'**, 24), text = **"Add Section"**, fg = **"Black"**,  
 bg = **"#A2ACEB"**)  
 txt\_title.pack()  
 txt\_code=Label(Forms, text = **"Code:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_code.grid(row = 0, stick = **"e"**)  
 txt\_result=Label(Buttons)  
 txt\_result.pack(side = TOP)  
  
 *# ==================================ENTRY WIDGET=======================================* cb\_code=ttk.Combobox(Forms, textvariable = CODE, width = 20)  
 cb\_code[**'values'**]=combobox\_input1()  
 cb\_code.grid(column = 1, row = 0)  
  
 *# ==================================BUTTONS WIDGET=====================================* btn\_create=Button(Buttons, width = 10, text = **"Create"**, command = Create)  
 btn\_create.pack(side = LEFT)  
 btn\_view=Button(Buttons, width = 10, text = **"View"**, command = View)  
 btn\_view.pack(side = LEFT)  
 btn\_update=Button(Buttons, width = 10, text = **"Update"**, state = DISABLED)  
 btn\_update.pack(side = LEFT)  
 btn\_delete=Button(Buttons, width = 10, text = **"Delete"**, command = Delete)  
 btn\_delete.pack(side = LEFT)  
 btn\_clear=Button(Buttons, width = 10, text = **"Clear"**, fg = **"Black"**, bg = **"#F2B6AE"**, command = Clear)  
 btn\_clear.pack(side = LEFT)  
 btn\_modify\_availability=Button(Top, width = 25, text = **"Modify Availability"**, font = (**'arial'**, 10, **'bold'**),  
 fg = **"Black"**, bg = **"#8e9ae6"**,  
 command = Modify\_Availability) *#* btn\_modify\_availability.pack(side = LEFT)  
 btn\_department=Button(Top, width = 25, text = **"Department"**, font = (**'arial'**, 10, **'bold'**),  
 fg = **"Black"**, bg = **"#a18ee6"**,  
 command = Department)  
 btn\_department.pack(side = LEFT)  
 btn\_refresh=Button(Top, width = 10, text = **"Refresh (F5)"**, font = (**'arial'**, 10, **'bold'**), fg = **"White"**, bg = **"green"**,  
 command = Refresh)  
 btn\_refresh.pack(side = RIGHT)  
  
 *# ==================================LIST WIDGET========================================* scrollbary=Scrollbar(Right, orient = VERTICAL)  
 scrollbarx=Scrollbar(Right, orient = HORIZONTAL)  
 columns=(**"Code"**,)  
 tree=ttk.Treeview(Right, columns = columns, selectmode = **"extended"**, height = 500,  
 yscrollcommand = scrollbary.set, xscrollcommand = scrollbarx.set)  
  
 *# Column Sorter* def treeview\_sort\_column(tree, col, reverse):  
 l = [(tree.set(k, col), k) for k in tree.get\_children(**''**)]  
 l.sort(reverse=reverse)  
  
 *# rearrange items in sorted positions* for index, (val, k) in enumerate(l):  
 tree.move(k, **''**, index)  
  
 *# reverse sort next time* tree.heading(col, command=lambda \_col=col: treeview\_sort\_column(tree, \_col, not reverse))  
  
 for col in columns:  
 tree.heading(col, text=col, command=lambda \_col=col: \  
 treeview\_sort\_column(tree, \_col, False))  
 *# for TREE Scroll Bars* scrollbary.config(command = tree.yview)  
 scrollbary.pack(side = RIGHT, fill = Y)  
 scrollbarx.config(command = tree.xview)  
 scrollbarx.pack(side = BOTTOM, fill = X)  
 *# inside the tree view* tree.heading(**"Code"**, text = **"Code"**, anchor = W)  
 tree.column(**'#0'**, stretch = NO, minwidth = 0, width = 0)  
 tree.bind(**'<ButtonRelease-1>'**, click)  
 tree.pack()  
  
 if \_\_name\_\_ == **'\_\_main\_\_'**:  
 View()  
 SC.mainloop()  
  
  
 def Subject():  
 subjroot=Toplevel(root)  
 subjroot.title(**"ECE-SPV"**)  
 subjroot.wm\_iconbitmap(**"ece-spv.ico"**)  
 screen\_width=subjroot.winfo\_screenwidth()  
 screen\_height=subjroot.winfo\_screenheight()  
 width=900  
 height=627  
 x=(screen\_width / 2) - (width / 2)  
 y=(screen\_height / 2) - (height / 2)  
 subjroot.geometry(**'%dx%d+%d+%d'** % (width, height, x, y))  
 subjroot.resizable(0, 0)  
  
 *# ==================================COMMANDS============================================* def click(event):  
 try:  
 selected=tree.focus() *# Grab record position/number* values=tree.item(selected, **'values'**) *# Grab record values* cb\_code.delete(0, **'end'**)  
 e\_description.delete(0, **'end'**)  
 e\_units.delete(0, **'end'**)  
 e\_hours.delete(0, **'end'**)  
 e\_compatibility.delete(0, **'end'**)  
 cb\_curriculum.delete(0, **'end'**)  
 e\_max\_numb\_of\_students.delete(0, **'end'**)  
 cb\_dept.delete(0, **'end'**)  
 cb\_code.insert(0, values[0])  
 e\_description.insert(0, values[1])  
 e\_units.insert(0, values[2])  
 e\_hours.insert(0, values[3])  
 e\_compatibility.insert(0, values[4])  
 cb\_curriculum.insert(0, values[5])  
 e\_max\_numb\_of\_students.insert(0, values[6])  
 cb\_dept.insert(0, values[7])  
 except IndexError: pass  
  
 def combobox\_input0():  
 ARADB()  
 cursor.execute(**"SELECT code FROM `subject`"**)  
 conn.commit()  
 data=[]  
 for row in cursor.fetchall():  
 data.append(row[0])  
 return data  
 cursor.close()  
 conn.close()  
  
 def combobox\_input1():  
 ARADB()  
 cursor.execute(**"SELECT code FROM `curriculum`"**)  
 conn.commit()  
 data=[]  
 for row in cursor.fetchall():  
 data.append(row[0])  
 return data  
 cursor.close()  
 conn.close()  
  
 def combobox\_input2():  
 ARADB()  
 cursor.execute(**"SELECT code FROM `dept`"**)  
 conn.commit()  
 data=[]  
 for row in cursor.fetchall():  
 data.append(row[0])  
 return data  
 cursor.close()  
 conn.close()  
  
 def Create():  
 if CODE.get() == **""** or DESCRIPTION.get() == **""** or UNITS.get() == 0 or HOURS.get() == 0 or UNITS.get() == **""** or HOURS.get() == **""** or CURRICULUM.get() == **""** or COMPATIBILITY.get() == **""** or MAX\_NUMB\_OF\_STUDENTS.get() == **""** or MAX\_NUMB\_OF\_STUDENTS.get() == 0 or DEPT.get() == **""**:  
 txt\_result.config(text = **"Please complete all the fields!"**, fg = **"red"**)  
 else:  
 ARADB()  
 cursor.execute(  
 **"INSERT INTO `subject` (code, description, units, numbHour, compatibility, curriculum, max\_numb\_of\_students, dept) VALUES(?, ?, ?, ?, ?, ?, ?, ?)"**,  
 (str(CODE.get()), str(DESCRIPTION.get()), int(UNITS.get()), int(HOURS.get()), str(COMPATIBILITY.get()),  
 str(CURRICULUM.get()), int(MAX\_NUMB\_OF\_STUDENTS.get()), str(DEPT.get())))  
 conn.commit()  
 CODE.set(**""**)  
 DESCRIPTION.set(**""**)  
 UNITS.set(**""**)  
 HOURS.set(**""**)  
 COMPATIBILITY.set(**""**)  
 CURRICULUM.set(**""**)  
 MAX\_NUMB\_OF\_STUDENTS.set(**""**)  
 DEPT.set(**""**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Created a data!"**, fg = **"green"**)  
  
 def View():  
 tree.delete(\*tree.get\_children())  
 ARADB()  
 cursor.execute(**"SELECT \* FROM `subject` ORDER BY `description` ASC"**)  
 fetch=cursor.fetchall()  
 for data in fetch:  
 tree.insert(**''**, **'end'**, values = (data[0], data[1], data[2], data[3], data[4], data[5], data[6], data[7]))  
 cursor.close()  
 conn.close()  
 txt\_result.config(text = **"Successfully viewed the data from database"**, fg = **"black"**)  
  
 def Update():  
 if CODE.get() == **""** or DESCRIPTION.get() == **""** or UNITS.get() == 0 or HOURS.get() == 0 or UNITS.get() == **""** or HOURS.get() == **""** or CURRICULUM.get() == **""** or COMPATIBILITY.get() == **""** or MAX\_NUMB\_OF\_STUDENTS.get() == **""** or MAX\_NUMB\_OF\_STUDENTS.get() == 0 or DEPT.get() == **""**:  
 txt\_result.config(text = **"Please complete all the fields!"**, fg = **"red"**)  
 else:  
 ARADB()  
 cursor.execute(  
 **"UPDATE `subject` SET description = :e\_description, units = :e\_units, numbHour = :e\_hours, compatibility = :e\_compatibility, curriculum = :cb\_curriculum, max\_numb\_of\_students = :e\_max\_numb\_of\_students, dept = :cb\_dept WHERE code = :cb\_code"""**,  
 {**'e\_description'**: e\_description.get(), **'e\_units'**: e\_units.get(), **'e\_hours'**: e\_hours.get(),  
 **'e\_compatibility'**: e\_compatibility.get(), **'cb\_curriculum'**: cb\_curriculum.get(),  
 **'e\_max\_numb\_of\_students'**: e\_max\_numb\_of\_students.get(), **'cb\_dept'**: cb\_dept.get(),  
 **'cb\_code'**: cb\_code.get()})  
 conn.commit()  
 cb\_code.delete(0, **'end'**)  
 e\_description.delete(0, **'end'**)  
 e\_units.delete(0, **'end'**)  
 e\_hours.delete(0, **'end'**)  
 e\_compatibility.delete(0, **'end'**)  
 cb\_curriculum.delete(0, **'end'**)  
 e\_max\_numb\_of\_students.delete(0, **'end'**)  
 cb\_dept.delete(0, **'end'**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Updated Successfully!"**, fg = **"green"**)  
  
 def Delete():  
 if CODE.get() == **""**:  
 txt\_result.config(text = **"Please enter the Subject Code!"**, fg = **"red"**)  
 else:  
 ARADB()  
 cursor.execute(**"DELETE FROM 'subject' WHERE code = '"** + str(CODE.get()) + **"'"**)  
 conn.commit()  
 cb\_code.delete(0, **'end'**)  
 e\_description.delete(0, **'end'**)  
 e\_units.delete(0, **'end'**)  
 e\_hours.delete(0, **'end'**)  
 e\_compatibility.delete(0, **'end'**)  
 cb\_curriculum.delete(0, **'end'**)  
 e\_max\_numb\_of\_students.delete(0, **'end'**)  
 cb\_dept.delete(0, **'end'**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Deleted Successfully!"**, fg = **"green"**)  
  
 def Clear():  
 result=messagebox.askquestion(**"Clear"**, **"This will clear the Section table.**\n**Do you want to proceed?"**, icon = **'warning'**)  
 if result == **'yes'**:  
 ARADB()  
 cursor.execute(**"""DROP TABLE subject"""**)  
 cursor.execute(**"""create table subject (code text PRIMARY KEY, description text, units FLOAT NOT NULL, numbHour FLOAT NOT NULL, compatibility text NOT NULL, curriculum text NOT NULL, max\_numb\_of\_students integer NOT NULL, dept text NOT NULL)"""**)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Table Cleared Successfully!"**, fg = **"green"**)  
 else:  
 pass  
  
 def Refresh():  
 subjroot.destroy()  
 Subject()  
  
 def Curriculum():  
 CC = Toplevel(root)  
 CC.title(**"ECE-SPV"**)  
 CC.wm\_iconbitmap(**"ece-spv.ico"**)  
 CC.screen\_width = CC.winfo\_screenwidth()  
 CC.screen\_height = CC.winfo\_screenheight()  
 CC.width = 720  
 CC.height = 627  
 CC.x = (screen\_width / 2) - (width / 2)  
 CC.y = (screen\_height / 2) - (height / 2)  
 CC.geometry(**'%dx%d+%d+%d'** % (width, height, x, y))  
 CC.resizable(0, 0)  
  
 *# ==================================COMMANDS============================================* def click(event):  
 try:  
 selected = tree.focus() *# Grab record position/number* values = tree.item(selected, **'values'**) *# Grab record values* cb\_code.delete(0, **'end'**)  
 cb\_code.insert(0, values[0])  
 except IndexError:  
 pass  
  
 def combobox\_input1():  
 ARADB()  
 cursor.execute(**"SELECT code FROM `curriculum`"**) *# SELECT code FROM `physical\_room`* conn.commit()  
 data = []  
 for row in cursor.fetchall():  
 data.append(row[0])  
 return data  
 cursor.close()  
 conn.close()  
  
 def Create():  
 if CODE.get() == **""**:  
 txt\_result.config(text = **"Please enter a Curriculum Code!"**, fg = **"red"**)  
 else:  
 ARADB()  
 cursor.execute(**"INSERT INTO `curriculum` (code) VALUES(?)"**, (str(CODE.get()),))  
 conn.commit()  
 CODE.set(**""**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Created a data!"**, fg = **"green"**)  
  
 def View():  
 tree.delete(\*tree.get\_children())  
 ARADB()  
 cursor.execute(**"SELECT \* FROM `curriculum` ORDER BY `code`"**)  
 fetch = cursor.fetchall()  
 for data in fetch:  
 tree.insert(**''**, **'end'**, values = (data[0]))  
 cursor.close()  
 conn.close()  
 txt\_result.config(text = **"Successfully viewed the data from database"**, fg = **"black"**)  
  
 def Delete():  
 if CODE.get() == **""**:  
 txt\_result.config(text = **"Please enter a Curriculum Code!"**, fg = **"red"**)  
 else:  
 ARADB()  
 cursor.execute(**"DELETE FROM 'curriculum' WHERE code = '"** + str(CODE.get()) + **"'"**)  
 conn.commit()  
 cb\_code.delete(0, **'end'**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Deleted Successfully!"**, fg = **"green"**)  
  
 def Clear():  
 result = messagebox.askquestion(**"Clear"**, **"This will clear the Section table.**\n**Do you want to proceed?"**,  
 icon = **'warning'**)  
 if result == **'yes'**:  
 ARADB()  
 cursor.execute(**"""DROP TABLE curriculum"""**)  
 cursor.execute(**"""create table curriculum (code text PRIMARY KEY)"""**)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Table Cleared Successfully!"**, fg = **"green"**)  
 else:  
 pass  
  
 def Refresh():  
 CC.destroy()  
 Curriculum()  
  
 *# ==================================VARIABLES==========================================* CODE = StringVar(CC)  
  
 *# ==================================FRAME==============================================* Top = Frame(CC, width = 900, height = 50, bd = 8, relief = **"raise"**, bg = **"#F7FAA1"**)  
 Top.pack(side = TOP)  
 Left = Frame(CC, width = 300, height = 500, bd = 8, relief = **"raise"**)  
 Left.pack(side = LEFT)  
 Right = Frame(CC, width = 600, height = 500, bd = 8, relief = **"raise"**)  
 Right.pack(side = RIGHT)  
 Forms = Frame(Left, width = 300, height = 450)  
 Forms.pack(side = TOP)  
 Buttons = Frame(Left, width = 300, height = 100, bd = 8, relief = **"raise"**)  
 Buttons.pack(side = BOTTOM)  
  
 *# ==================================LABEL WIDGET=======================================* txt\_title = Label(Top, width = 900, font = (**'arial'**, 24), text = **"Curriculum"**, fg = **"Black"**,  
 bg = **"#FEF44E"**)  
 txt\_title.pack()  
 txt\_code = Label(Forms, text = **"Curriculum:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_code.grid(row = 0, stick = **"e"**)  
 txt\_result = Label(Buttons)  
 txt\_result.pack(side = TOP)  
  
 *# ==================================ENTRY WIDGET=======================================* cb\_code = ttk.Combobox(Forms, textvariable = CODE, width = 20)  
 cb\_code[**'values'**] = combobox\_input1()  
 cb\_code.grid(column = 1, row = 0)  
  
 *# ==================================BUTTONS WIDGET=====================================* btn\_create = Button(Buttons, width = 10, text = **"Create"**, command = Create)  
 btn\_create.pack(side = LEFT)  
 btn\_view = Button(Buttons, width = 10, text = **"View"**, command = View)  
 btn\_view.pack(side = LEFT)  
 btn\_update = Button(Buttons, width = 10, text = **"Update"**, state = DISABLED)  
 btn\_update.pack(side = LEFT)  
 btn\_delete = Button(Buttons, width = 10, text = **"Delete"**, command = Delete)  
 btn\_delete.pack(side = LEFT)  
 btn\_clear = Button(Buttons, width = 10, text = **"Clear"**, fg = **"Black"**, bg = **"#F2B6AE"**, command = Clear)  
 btn\_clear.pack(side = LEFT)  
 btn\_refresh = Button(Top, width = 10, text = **"Refresh (F5)"**, font = (**'arial'**, 10, **'bold'**), fg = **"White"**,  
 bg = **"green"**, command = Refresh)  
 btn\_refresh.pack(side = RIGHT)  
  
 *# ==================================LIST WIDGET========================================* scrollbary = Scrollbar(Right, orient = VERTICAL)  
 scrollbarx = Scrollbar(Right, orient = HORIZONTAL)  
 columns = (**"Curriculum"**,)  
 tree = ttk.Treeview(Right, columns = columns, selectmode = **"extended"**, height = 500,  
 yscrollcommand = scrollbary.set, xscrollcommand = scrollbarx.set)  
  
 *# Column Sorter* def treeview\_sort\_column(tree, col, reverse):  
 l = [(tree.set(k, col), k) for k in tree.get\_children(**''**)]  
 l.sort(reverse = reverse)  
  
 *# rearrange items in sorted positions* for index, (val, k) in enumerate(l):  
 tree.move(k, **''**, index)  
  
 *# reverse sort next time* tree.heading(col, command = lambda \_col=col: treeview\_sort\_column(tree, \_col, not reverse))  
  
 for col in columns:  
 tree.heading(col, text = col, command = lambda \_col=col: \  
 treeview\_sort\_column(tree, \_col, False))  
 *# for TREE Scroll Bars* scrollbary.config(command = tree.yview)  
 scrollbary.pack(side = RIGHT, fill = Y)  
 scrollbarx.config(command = tree.xview)  
 scrollbarx.pack(side = BOTTOM, fill = X)  
 *# inside the tree view* tree.heading(**"Curriculum"**, text = **"Curriculum"**, anchor = W)  
 tree.column(**'#0'**, stretch = NO, minwidth = 0, width = 0)  
 tree.bind(**'<ButtonRelease-1>'**, click)  
 tree.pack()  
  
 if \_\_name\_\_ == **'\_\_main\_\_'**:  
 View()  
 CC.mainloop()  
  
 *# ==================================VARIABLES==========================================* CODE=StringVar(subjroot)  
 DESCRIPTION=StringVar(subjroot)  
 UNITS=IntVar(subjroot)  
 HOURS=IntVar(subjroot)  
 COMPATIBILITY=StringVar(subjroot)  
 CURRICULUM=StringVar(subjroot)  
 MAX\_NUMB\_OF\_STUDENTS=IntVar(subjroot)  
 DEPT=StringVar(subjroot)  
  
 *# ==================================FRAME==============================================* Top=Frame(subjroot, width = 900, height = 50, bd = 8, relief = **"raise"**, bg = **"#F7FAA1"**)  
 Top.pack(side = TOP)  
 Left=Frame(subjroot, width = 300, height = 500, bd = 8, relief = **"raise"**)  
 Left.pack(side = LEFT)  
 Right=Frame(subjroot, width = 600, height = 500, bd = 8, relief = **"raise"**)  
 Right.pack(side = RIGHT)  
 Forms=Frame(Left, width = 300, height = 450)  
 Forms.pack(side = TOP)  
 Buttons=Frame(Left, width = 300, height = 100, bd = 8, relief = **"raise"**)  
 Buttons.pack(side = BOTTOM)  
 CheckBox=Frame(Forms)  
 VR=Checkbutton(CheckBox, text = **"YES!"**, variable = COMPATIBILITY, onvalue = **"VR"**, offvalue = **"PR"**,  
 font = (**'arial'**, 8)).pack(  
 side = LEFT)  
  
 *# ==================================LABEL WIDGET=======================================* txt\_title=Label(Top, width = 900, font = (**'arial'**, 24), text = **"Subject"**, fg = **"Black"**, bg = **"#FDFD97"**)  
 txt\_title.pack()  
 txt\_code=Label(Forms, text = **"Code:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_code.grid(row = 0, stick = **"e"**)  
 txt\_description=Label(Forms, text = **"Description:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_description.grid(row = 1, stick = **"e"**)  
 txt\_units=Label(Forms, text = **"Units:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_units.grid(row = 2, stick = **"e"**)  
 txt\_hours=Label(Forms, text = **"Hours:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_hours.grid(row = 3, stick = **"e"**)  
 txt\_compatibility=Label(Forms, text = **"Is this a Lecture Subject?"**, font = (**'arial'**, 16), bd = 15)  
 txt\_compatibility.grid(row = 4, stick = **"e"**)  
 txt\_curriculum=Label(Forms, text = **"Curriculum:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_curriculum.grid(row = 5, stick = **"e"**)  
 txt\_max\_numb\_of\_students=Label(Forms, text = **"Students Max #:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_max\_numb\_of\_students.grid(row = 6, stick = **"e"**)  
 txt\_dept=Label(Forms, text = **"Select Dept:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_dept.grid(row = 7, stick = **"e"**)  
 txt\_result=Label(Buttons)  
 txt\_result.pack(side = TOP)  
  
 *# ==================================ENTRY WIDGET=======================================* cb\_code=ttk.Combobox(Forms, textvariable = CODE, width = 20)  
 cb\_code[**'values'**]=combobox\_input0()  
 cb\_code.grid(row = 0, column = 1)  
 e\_description=Entry(Forms, textvariable = DESCRIPTION, width = 20)  
 e\_description.grid(row = 1, column = 1)  
 e\_units=Entry(Forms, textvariable = UNITS, width = 20)  
 e\_units.grid(row = 2, column = 1)  
 e\_hours=Entry(Forms, textvariable = HOURS, width = 20)  
 e\_hours.grid(row = 3, column = 1)  
 e\_compatibility=Entry(Forms, textvariable = COMPATIBILITY, width = 20)  
 CheckBox.grid(row = 4, column = 1)  
 cb\_curriculum=ttk.Combobox(Forms, textvariable = CURRICULUM, width = 20)  
 cb\_curriculum[**'values'**]=combobox\_input1()  
 cb\_curriculum.grid(row = 5, column = 1)  
 e\_max\_numb\_of\_students=Entry(Forms, textvariable = MAX\_NUMB\_OF\_STUDENTS, width = 20)  
 e\_max\_numb\_of\_students.grid(row = 6, column = 1)  
 cb\_dept=ttk.Combobox(Forms, textvariable = DEPT, width = 20)  
 cb\_dept[**'values'**]=combobox\_input2()  
 cb\_dept.grid(row = 7, column = 1)  
  
 *# ==================================BUTTONS WIDGET=====================================* btn\_create=Button(Buttons, width = 10, text = **"Create"**, command = Create)  
 btn\_create.pack(side = LEFT)  
 btn\_view=Button(Buttons, width = 10, text = **"View"**, command = View)  
 btn\_view.pack(side = LEFT)  
 btn\_update=Button(Buttons, width = 10, text = **"Update"**, command = Update)  
 btn\_update.pack(side = LEFT)  
 btn\_delete=Button(Buttons, width = 10, text = **"Delete"**, command = Delete)  
 btn\_delete.pack(side = LEFT)  
 btn\_clear=Button(Buttons, width = 10, text = **"Clear"**, fg = **"Black"**, bg = **"#F2B6AE"**, command = Clear)  
 btn\_clear.pack(side = LEFT)  
 btn\_curriculum=Button(Top, width = 25, text = **"Curriculum"**, font = (**'arial'**, 10, **'bold'**),  
 fg = **"Black"**, bg = **"#FEF44E"**,  
 command = Curriculum)   
 btn\_curriculum.pack(side = LEFT)  
 btn\_refresh=Button(Top, width = 10, text = **"Refresh (F5)"**, font = (**'arial'**, 10, **'bold'**), fg = **"white"**, bg = **"green"**,  
 command = Refresh)  
 btn\_refresh.pack(side = RIGHT)  
  
 *# ==================================LIST WIDGET========================================* scrollbary=Scrollbar(Right, orient = VERTICAL)  
 scrollbarx=Scrollbar(Right, orient = HORIZONTAL)  
 columns=(**"Code"**, **"Description"**, **"Units"**, **"Hours #"**, **"Compatibility"**, **"Curriculum"**, **"Students #"**, **"Dept"**)  
 tree=ttk.Treeview(Right, columns = columns, selectmode = **"extended"**, height = 500,  
 yscrollcommand = scrollbary.set, xscrollcommand = scrollbarx.set)  
  
 def treeview\_sort\_column(tree, col, reverse):  
 l=[(tree.set(k, col), k) for k in tree.get\_children(**''**)]  
 l.sort(reverse = reverse)  
  
 *# rearrange items in sorted positions* for index, (val, k) in enumerate(l):  
 tree.move(k, **''**, index)  
  
 *# reverse sort next time* tree.heading(col, command = lambda \_col=col: treeview\_sort\_column(tree, \_col, not reverse))  
  
 for col in columns:  
 tree.heading(col, text = col, command = lambda \_col=col: \  
 treeview\_sort\_column(tree, \_col, False))  
 *# for TREE Scroll Bars* scrollbary.config(command = tree.yview)  
 scrollbary.pack(side = RIGHT, fill = Y)  
 scrollbarx.config(command = tree.xview)  
 scrollbarx.pack(side = BOTTOM, fill = X)  
 *# inside the tree view* tree.heading(**'Code'**, text = **"Code"**, anchor = W)  
 tree.heading(**'Description'**, text = **"Description"**, anchor = W)  
 tree.heading(**'Units'**, text = **"Units"**, anchor = W)  
 tree.heading(**'Hours #'**, text = **"Hours #"**, anchor = W)  
 tree.heading(**'Compatibility'**, text = **"Compatibility"**, anchor = W)  
 tree.heading(**'Curriculum'**, text = **"Curriculum"**, anchor = W)  
 tree.heading(**'Students #'**, text = **"Students #"**, anchor = W)  
 tree.heading(**'Dept'**, text = **"Dept"**, anchor = W)  
 tree.column(**'#0'**, stretch = NO, minwidth = 0, width = 0)  
 tree.column(**'#1'**, stretch = NO, minwidth = 90, width = 90)  
 tree.column(**'#2'**, stretch = NO, minwidth = 140, width = 140)  
 tree.column(**'#3'**, stretch = NO, minwidth = 40, width = 40)  
 tree.column(**'#4'**, stretch = NO, minwidth = 55, width = 55)  
 tree.column(**'#5'**, stretch = NO, minwidth = 40, width = 40)  
 tree.column(**'#6'**, stretch = NO, minwidth = 50, width = 50)  
 tree.column(**'#7'**, stretch = NO, minwidth = 65, width = 65)  
 tree.bind(**'<ButtonRelease-1>'**, click)  
 tree.pack()  
  
 *# ==================================INITIALIZATION=====================================* if \_\_name\_\_ == **'\_\_main\_\_'**:  
 View()  
 subjroot.mainloop()  
  
  
 def Room():  
 roomroot=Toplevel(root)  
 roomroot.title(**"ECE-SPV"**)  
 roomroot.wm\_iconbitmap(**"ece-spv.ico"**)  
 screen\_width=roomroot.winfo\_screenwidth()  
 screen\_height=roomroot.winfo\_screenheight()  
 width=900  
 height=627  
 x=(screen\_width / 2) - (width / 2)  
 y=(screen\_height / 2) - (height / 2)  
 roomroot.geometry(**'%dx%d+%d+%d'** % (width, height, x, y))  
 roomroot.resizable(0, 0)  
  
 *# ==================================COMMANDS============================================* def click(event):  
 try:  
 selected=tree.focus() *# Grab record position/number* values=tree.item(selected, **'values'**) *# Grab record values* cb\_number.delete(0, **'end'**)  
 e\_capacity.delete(0, **'end'**)  
 e\_type.delete(0, **'end'**)  
 cb\_dept.delete(0, **'end'**)  
 cb\_number.insert(0, values[0])  
 e\_capacity.insert(0, values[1])  
 e\_type.insert(0, values[2])  
 cb\_dept.insert(0, values[3])  
 except IndexError: pass  
  
 def combobox\_input0():  
 ARADB()  
 cursor.execute(**"SELECT number FROM `room`"**)  
 conn.commit()  
 data=[]  
 for row in cursor.fetchall():  
 data.append(row[0])  
 return data  
 cursor.close()  
 conn.close()  
  
 def combobox\_input1():  
 ARADB()  
 cursor.execute(**"SELECT code FROM `dept`"**)  
 conn.commit()  
 data=[]  
 for row in cursor.fetchall():  
 data.append(row[0])  
 return data  
 cursor.close()  
 conn.close()  
  
 def Create():  
 if NUMBER.get() == **""** or CAPACITY.get() == 0 or CAPACITY.get() == **""** or TYPE.get() == **""** or DEPT.get() == **""**:  
 txt\_result.config(text = **"Please complete all fields!"**, fg = **"red"**)  
 else:  
 ARADB()  
 cursor.execute(**"INSERT INTO `room` (number, capacity, type, dept) VALUES(?, ?, ?, ?)"**,  
 (str(NUMBER.get()), int(CAPACITY.get()), str(TYPE.get()), str(DEPT.get()),))  
 conn.commit()  
 NUMBER.set(**""**)  
 CAPACITY.set(**""**)  
 TYPE.set(**""**)  
 DEPT.set(**""**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Created a data!"**, fg = **"green"**)  
  
 def View():  
 tree.delete(\*tree.get\_children())  
 ARADB()  
 cursor.execute(**"SELECT \* FROM `room` ORDER BY `type` ASC"**)  
 fetch=cursor.fetchall()  
 for data in fetch:  
 tree.insert(**''**, **'end'**, values = (data[0], data[1], data[2], data[3]))  
 cursor.close()  
 conn.close()  
 txt\_result.config(text = **"Successfully viewed the data from database"**, fg = **"black"**)  
  
 def Update():  
 if NUMBER.get() == **""** or CAPACITY.get() == 0 or CAPACITY.get() == **""** or TYPE.get() == **""** or DEPT.get() == **""**:  
 txt\_result.config(text = **"Please complete all fields!"**, fg = **"red"**)  
 else:  
 ARADB()  
 cursor.execute(  
 **"UPDATE `room` SET capacity = :e\_capacity, type = :e\_type, dept = :cb\_dept WHERE number = :cb\_number"""**,  
 {  
 **'e\_capacity'**: e\_capacity.get(), **'e\_type'**: e\_type.get(), **'cb\_dept'**: cb\_dept.get(),  
 **'cb\_number'**: cb\_number.get()})  
 conn.commit()  
 cb\_number.delete(0, **'end'**)  
 e\_capacity.delete(0, **'end'**)  
 e\_type.delete(0, **'end'**)  
 cb\_dept.delete(0, **'end'**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Updated Successfully!"**, fg = **"green"**)  
  
 def Delete():  
 if NUMBER.get() == **""**:  
 txt\_result.config(text = **"Please enter a Room Number!"**, fg = **"red"**)  
 else:  
 ARADB()  
 cursor.execute(**"DELETE FROM 'room' WHERE number = '"** + str(NUMBER.get()) + **"'"**)  
 conn.commit()  
 cb\_number.delete(0, **'end'**)  
 e\_capacity.delete(0, **'end'**)  
 e\_type.delete(0, **'end'**)  
 cb\_dept.delete(0, **'end'**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Deleted Successfully!"**, fg = **"green"**)  
  
 def Clear():  
 result=messagebox.askquestion(**"Clear"**, **"This will clear the Room table.**\n**Do you want to proceed?"**, icon = **'warning'**)  
 if result == **'yes'**:  
 ARADB()  
 cursor.execute(**"""DROP TABLE room"""**)  
 cursor.execute(**"""create table room (number text PRIMARY KEY, capacity integer NOT NULL, type text NOT NULL, dept text NOT NULL)"""**)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Table Cleared Successfully!"**, fg = **"green"**)  
 else:  
 pass  
  
 def Modify\_Availability():  
 RA=Toplevel(roomroot)  
 RA.title(**"ECE-SPV"**)  
 RA.wm\_iconbitmap(**"ece-spv.ico"**)  
 RA.screen\_width=RA.winfo\_screenwidth()  
 RA.screen\_height=RA.winfo\_screenheight()  
 RA.width=720  
 RA.height=627  
 RA.x=(screen\_width / 2) - (width / 2)  
 RA.y=(screen\_height / 2) - (height / 2)  
 RA.geometry(**'%dx%d+%d+%d'** % (width, height, x, y))  
 RA.resizable(0, 0)  
  
 *# ==================================COMMANDS============================================* class DBMgr:  
 def \_\_init\_\_(self):  
 self.\_conn=sqlite.connect(**'ARADB.db'**) *# sql connector* self.\_cursor=self.\_conn.cursor() *# sql cursor* self.\_times=self.\_select\_times() *# select time sql cursor* self.\_timepoints=self.\_select\_timepoints() *# select timepoints sql cursor* self.\_meetingTimes=self.\_select\_meeting\_times() *# select meeting time sql cursor* self.\_instructors=self.\_select\_instructors() *# select instructor ql cursor* self.\_subjects=self.\_select\_subjects() *# select subject sql cursor* self.\_rooms=self.\_select\_rooms() *# select room sql cursor* self.\_sections=self.\_select\_sections() *# select section sql cursor* self.\_numberOfClasses=0 *# initial number of classes* for i in range(0, len(self.\_sections)):  
 self.\_numberOfClasses+=len(  
 self.\_sections[i].get\_subjects()) *# auto-itereation of number of classes creation  
  
 # select section sql command* def \_select\_sections(self):  
 self.\_cursor.execute(**"SELECT \* FROM section"**)  
 sections=self.\_cursor.fetchall()  
 returnSections=[]  
 for i in range(0, len(sections)):  
 returnSections.append(Section(sections[i][0], self.\_select\_section\_subjects(sections[i][0]),  
 self.\_select\_section\_availability(sections[i][0])))  
 return returnSections  
  
 *# select subject sql command* def \_select\_subjects(self):  
 self.\_cursor.execute(**"SELECT \* FROM subject"**)  
 subjects=self.\_cursor.fetchall()  
 returnSubjects=[]  
 for i in range(0, len(subjects)):  
 returnSubjects.append(  
 Subject(subjects[i][0], subjects[i][1], subjects[i][2], subjects[i][3], subjects[i][4],  
 subjects[i][5], subjects[i][6], subjects[i][7], self.\_select\_subject\_instructors(  
 subjects[i][0])))   
 return returnSubjects  
  
 *# select instructor sql command* def \_select\_instructors(self):  
 self.\_cursor.execute(**"SELECT \* FROM instructor"**)  
 instructors=self.\_cursor.fetchall()  
 returnInstructors=[]  
 for i in range(0, len(instructors)):  
 returnInstructors.append(  
 Instructor(instructors[i][0], instructors[i][1], instructors[i][2], instructors[i][3],  
 self.\_select\_instructor\_availability(instructors[i][0]),  
 self.\_select\_instructor\_availability\_start(instructors[i][0]),  
 self.\_select\_instructor\_availability\_end(instructors[i][0]),  
 self.\_select\_instructor\_overtime(instructors[i][0])))  
 return returnInstructors  
  
 *# select room sql command* def \_select\_rooms(self):  
 self.\_cursor.execute(**"SELECT \* FROM room"**)  
 rooms=self.\_cursor.fetchall()  
 returnRooms=[]  
 for i in range(0, len(rooms)):  
 returnRooms.append(Room(rooms[i][0], rooms[i][1], rooms[i][2], rooms[i][3],  
 self.\_select\_room\_availability(  
 rooms[i][  
 0])))   
 return returnRooms  
  
 *# select meeting time sql command* def \_select\_meeting\_times(self):  
 self.\_cursor.execute(**"SELECT \* FROM meeting\_time"**)  
 meetingTimes=self.\_cursor.fetchall()  
 returnMeetingTimes=[]  
 for i in range(0, len(meetingTimes)):  
 returnMeetingTimes.append(  
 MeetingTime(meetingTimes[i][0], meetingTimes[i][1], meetingTimes[i][2],  
 meetingTimes[i][3], meetingTimes[i][4],  
 self.\_select\_rest\_time(meetingTimes[i][0]),  
 self.\_select\_break\_time(meetingTimes[i][0])))   
 return returnMeetingTimes  
  
 *# select time sql command* def \_select\_times(self):  
 self.\_cursor.execute(**"SELECT \* FROM time"**)  
 times=self.\_cursor.fetchall()  
 returnTimes=[]  
 for i in range(0, len(times)):  
 returnTimes.append(Time(times[i][0]))  
 return returnTimes  
  
 *# select timepoint sql command* def \_select\_timepoints(self):  
 self.\_cursor.execute(**"SELECT \* FROM time\_point"**)  
 timepoints=self.\_cursor.fetchall()  
 returnTimePoints=[]  
 for i in range(0, len(timepoints)):  
 returnTimePoints.append(TimePoint(timepoints[i][0],  
 self.\_select\_point\_block(timepoints[i][0]),  
 self.\_select\_phantom\_block(timepoints[i][0])))  
 return returnTimePoints  
  
 *# GETTING VALUE* def \_select\_section\_subjects(self, sectionCode):  
 self.\_cursor.execute(**"SELECT \* FROM section\_subject where section\_code == '"** + sectionCode + **"'"**)  
 dbSubjectCodes=self.\_cursor.fetchall()  
 subjectCodes=[]  
 for i in range(0, len(dbSubjectCodes)):  
 subjectCodes.append(dbSubjectCodes[i][1])  
 returnValue=[]  
 for i in range(0, len(self.\_subjects)):  
 if self.\_subjects[i].get\_code() in subjectCodes:  
 returnValue.append(self.\_subjects[i])  
 return returnValue  
  
 def \_select\_subject\_instructors(self, subjectCode):  
 self.\_cursor.execute(**"SELECT \* FROM subject\_instructor where subject\_code == '"** + subjectCode + **"'"**)  
 dbInstructorNumbers=self.\_cursor.fetchall()  
 instructorNumbers=[]  
 for i in range(0, len(dbInstructorNumbers)): instructorNumbers.append(dbInstructorNumbers[i][1])  
 returnValue=[]  
 for i in range(0, len(self.\_instructors)):  
 if self.\_instructors[i].get\_number() in instructorNumbers:  
 returnValue.append(self.\_instructors[i])  
 return returnValue  
  
 def \_select\_instructor\_availability(self, instructor):  
 self.\_cursor.execute(  
 **"SELECT \* from instructor\_availability where instructor\_id == '"** + instructor + **"'"**)  
 instructorMTsRS=self.\_cursor.fetchall()  
 instructorMTs=[]  
 for i in range(0, len(instructorMTsRS)):  
 instructorMTs.append(instructorMTsRS[i][1])  
 instructorAvailability=list()  
 for i in range(0, len(self.\_meetingTimes)):  
 if self.\_meetingTimes[i].get\_id() in instructorMTs:  
 instructorAvailability.append(self.\_meetingTimes[i].get\_id())  
 return instructorAvailability  
  
 def \_select\_instructor\_availability\_start(self, instructor):  
 self.\_cursor.execute(  
 **"SELECT \* from instructor\_availability where instructor\_id == '"** + instructor + **"'"**)  
 instructorMTsRS=self.\_cursor.fetchall()  
 instructorMTsSTART=[]  
 for i in range(0, len(instructorMTsRS)):  
 instructorMTsSTART.append(instructorMTsRS[i][2])  
 instructorAvailabilitySTART=list()  
 for i in range(0, len(self.\_timepoints)):  
 if self.\_timepoints[i].get\_point() in instructorMTsSTART:  
 instructorAvailabilitySTART.append(self.\_timepoints[i].get\_point())  
 return instructorAvailabilitySTART  
  
 def \_select\_instructor\_availability\_end(self, instructor):  
 self.\_cursor.execute(  
 **"SELECT \* from instructor\_availability where instructor\_id == '"** + instructor + **"'"**)  
 instructorMTsRS=self.\_cursor.fetchall()  
 instructorMTsEND=[]  
 for i in range(0, len(instructorMTsRS)):  
 instructorMTsEND.append(instructorMTsRS[i][3])  
 instructorAvailabilityEND=list()  
 for i in range(0, len(self.\_timepoints)):  
 if self.\_timepoints[i].get\_point() in instructorMTsEND:  
 instructorAvailabilityEND.append(self.\_timepoints[i].get\_point())  
 return instructorAvailabilityEND  
  
 def \_select\_instructor\_overtime(self, instructor):  
 self.\_cursor.execute(  
 **"SELECT \* from instructor\_overtime where instructor\_id == '"** + instructor + **"'"**)  
 instructorMTsRS=self.\_cursor.fetchall()  
 instructorMTs=[]  
 for i in range(0, len(instructorMTsRS)):  
 instructorMTs.append(instructorMTsRS[i][1])  
 instructorOvertime=list()  
 for i in range(0, len(self.\_meetingTimes)):  
 if self.\_meetingTimes[i].get\_id() in instructorMTs:  
 instructorOvertime.append(self.\_meetingTimes[i].get\_id())  
 return instructorOvertime  
  
 def \_select\_room\_availability(self, room):  
 self.\_cursor.execute(**"SELECT \* from room\_availability where room\_number = '"** + room + **"'"**)  
 roomMTsRS=self.\_cursor.fetchall()  
 roomMTs=[]  
 for i in range(0, len(roomMTsRS)): roomMTs.append(roomMTsRS[i][1])  
 roomAvailability=list()  
 for i in range(0, len(self.\_meetingTimes)):  
 if self.\_meetingTimes[i].get\_id() in roomMTs:  
 roomAvailability.append(self.\_meetingTimes[i].get\_id())  
 return roomAvailability  
  
 def \_select\_section\_availability(self, section):  
 self.\_cursor.execute(**"SELECT \* from section\_availability where section\_code = '"** + section + **"'"**)  
 sectionMTsRS=self.\_cursor.fetchall()  
 sectionMTs=[]  
 for i in range(0, len(sectionMTsRS)): sectionMTs.append(sectionMTsRS[i][1])  
 sectionAvailability=list()  
 for i in range(0, len(self.\_meetingTimes)):  
 if self.\_meetingTimes[i].get\_id() in sectionMTs:  
 sectionAvailability.append(self.\_meetingTimes[i].get\_id())  
 return sectionAvailability  
  
 def \_select\_rest\_time(self, meeting\_time):  
 self.\_cursor.execute(**"SELECT \* from rest\_time where meeting\_time\_id = '"** + meeting\_time + **"'"**)  
 timeMTsRS=self.\_cursor.fetchall()  
 timeMTs=[]  
 for i in range(0, len(timeMTsRS)): timeMTs.append(timeMTsRS[i][1])  
 restTime=list()  
 for i in range(0, len(self.\_times)):  
 if self.\_times[i].get\_block() in timeMTs:  
 restTime.append(self.\_times[i].get\_block())  
 return restTime  
  
 def \_select\_break\_time(self, meeting\_time):  
 self.\_cursor.execute(**"SELECT \* from break\_time where meeting\_time\_id = '"** + meeting\_time + **"'"**)  
 timeMTsRS=self.\_cursor.fetchall()  
 timeMTs=[]  
 for i in range(0, len(timeMTsRS)): timeMTs.append(timeMTsRS[i][1])  
 breakTime=list()  
 for i in range(0, len(self.\_times)):  
 if self.\_times[i].get\_block() in timeMTs:  
 breakTime.append(self.\_times[i].get\_block())  
 return breakTime  
  
 def \_select\_point\_block(self, point):  
 self.\_cursor.execute(**"SELECT \* from point\_block where time\_point == '"** + point + **"'"**)  
 pointBlocksRS=self.\_cursor.fetchall()  
 pointBlocks=[]  
 for i in range(0, len(pointBlocksRS)): pointBlocks.append(pointBlocksRS[i][1])  
 returnValue=list()  
 for i in range(0, len(self.\_times)):  
 if self.\_times[i].get\_block() in pointBlocks:  
 returnValue.append(self.\_times[i].get\_block())  
 return returnValue  
  
 def \_select\_phantom\_block(self, point):  
 self.\_cursor.execute(**"SELECT \* from phantom\_block where time\_point == '"** + point + **"'"**)  
 pointBlocksRS=self.\_cursor.fetchall()  
 pointBlocks=[]  
 for i in range(0, len(pointBlocksRS)): pointBlocks.append(pointBlocksRS[i][1])  
 returnValue=list()  
 for i in range(0, len(self.\_times)):  
 if self.\_times[i].get\_block() in pointBlocks:  
 returnValue.append(self.\_times[i].get\_block())  
 return returnValue  
  
 def get\_sections(self):  
 return self.\_sections *# get item to section table* def get\_subjects(self):  
 return self.\_subjects *# get item to subject table* def get\_instructors(self):  
 return self.\_instructors *# get item to instructor table* def get\_meetingTimes(self):  
 return self.\_meetingTimes *# get item to meeting time table* def get\_times(self):  
 return self.\_times *# get item to time table* def get\_timepoints(self):  
 return self.\_timepoints *# get item to time table* def get\_rooms(self):  
 return self.\_rooms *# get item to room table* def get\_numberOfClasses(self):  
 return self.\_numberOfClasses *# get item to number of class table* class Schedule:  
 def \_\_init\_\_(self):  
 self.\_data=dbMgr  
 self.\_classes=[]  
 self.\_conflicts=[]  
 self.\_fitness=-1  
 self.\_classNumb=0  
 self.\_isFitnessChanged=True  
 self.\_generationNumber=0  
  
 def get\_classes(self):  
 self.\_isFitnessChanged=True  
 return self.\_classes  
  
 def get\_conflicts(self):  
 return self.\_conflicts  
  
 def get\_fitness(self):  
 if (self.\_isFitnessChanged == True):  
 self.\_fitness=self.calculate\_fitness()  
 self.\_isFitnessChanged=False  
 return self.\_fitness  
  
 def initialize(self):  
 sections=dbMgr.get\_sections() *# point to section table* meetingTimes=dbMgr.get\_meetingTimes()  
 rooms=dbMgr.get\_rooms()  
 classes=self.\_classes  
 PR=list()  
 instructorList=list()  
 MT1=list()  
 MT1p5=list()  
 MT2=list()  
 MT3=list()  
 MT4=list()  
 MT1v2=list()  
 MT1p5v2=list()  
 MT2v2=list()  
 MT3v2=list()  
 for RM in range(0, len(rooms)):  
 if (rooms[RM].get\_type() == **'PR'**):  
 PR.append(rooms[RM])  
 for MT in range(0, len(meetingTimes)):  
 if (meetingTimes[MT].get\_MThour() == 1):  
 MT1.append(meetingTimes[MT])  
 MT1v2.append(meetingTimes[MT])  
 if (meetingTimes[MT].get\_MThour() == 1.5):  
 MT1p5.append(meetingTimes[MT])  
 MT1p5v2.append(meetingTimes[MT])  
 if (meetingTimes[MT].get\_MThour() == 2):  
 MT2.append(meetingTimes[MT])  
 MT2v2.append(meetingTimes[MT])  
 if (meetingTimes[MT].get\_MThour() == 3):  
 MT3.append(meetingTimes[MT])  
 MT3v2.append(meetingTimes[MT])  
 if (meetingTimes[MT].get\_MThour() == 4):  
 MT4.append(meetingTimes[MT])  
 for i in range(0, len(sections)): *# List all the items per coloumn* subjects=sections[i].get\_subjects() *# point to subject table* for j in range(0, len(subjects)): *# List all the items per coloumn* newClass=Class(self.\_classNumb, sections[i], subjects[j]) *# Create a new class* self.\_classNumb+=1 *# Iterate the class creation* newClass.set\_instructor(subjects[j].get\_instructors()[rnd.randrange(0, len(subjects[  
 j].get\_instructors()))]) *# Pick a random instructor taht can teach the assigned subject for the new class* if (subjects[j].get\_compatibility() == **'PR'**):  
 newClass.set\_room(  
 PR[rnd.randrange(0, len(PR))]) *# Pick a random room in database for the new class* else:  
 newClass.set\_room(  
 rooms[rnd.randrange(0,  
 len(rooms))]) *# Pick a random room in database for the new class"""* def wholetwohours():  
 newClass.set\_meetingTime(MT2[rnd.randrange(0, len(MT2))])  
 newClass.set\_meetingTime1(dbMgr.get\_meetingTimes()[0])  
  
 def wholethreehours():  
 newClass.set\_meetingTime(MT3[rnd.randrange(0, len(MT3))])  
 newClass.set\_meetingTime1(dbMgr.get\_meetingTimes()[0])  
  
 def wholefourhours():  
 newClass.set\_meetingTime(MT4[rnd.randrange(0, len(MT4))])  
 newClass.set\_meetingTime1(dbMgr.get\_meetingTimes()[0])  
  
 def splittwohours():  
 newClass.set\_meetingTime(MT1[rnd.randrange(0, len(MT1))])  
 newClass.set\_meetingTime1(MT1v2[rnd.randrange(0, len(MT1v2))])  
  
 def splitthreehours():  
 newClass.set\_meetingTime(MT1p5[rnd.randrange(0, len(MT1p5))])  
 newClass.set\_meetingTime1(MT1p5v2[rnd.randrange(0, len(MT1p5v2))])  
  
 def splitfourhours():  
 newClass.set\_meetingTime(MT2[rnd.randrange(0, len(MT2))])  
 newClass.set\_meetingTime1(MT2v2[rnd.randrange(0, len(MT2v2))])  
  
 def splitsixhours():  
 newClass.set\_meetingTime(MT3[rnd.randrange(0, len(MT3))])  
 newClass.set\_meetingTime1(MT3v2[rnd.randrange(0, len(MT3v2))])  
  
 def splittwoplusone():  
 newClass.set\_meetingTime(MT2[rnd.randrange(0, len(MT2))])  
 newClass.set\_meetingTime1(MT1[rnd.randrange(0, len(MT1))])  
  
 def splitthreeplusone():  
 newClass.set\_meetingTime(MT3[rnd.randrange(0, len(MT3))])  
 newClass.set\_meetingTime1(MT1[rnd.randrange(0, len(MT1))])  
  
 def splitfourplustwo():  
 newClass.set\_meetingTime(MT4[rnd.randrange(0, len(MT4))])  
 newClass.set\_meetingTime1(MT2[rnd.randrange(0, len(MT2))])  
  
 def call\_funcs\_randomly(funcs):  
 shuffle(funcs)  
 for func in funcs:  
 func()  
  
 if DISABLE\_UNEQUAL\_SPLIT == True:  
 if (subjects[j].get\_numbHour() == 1):  
 newClass.set\_meetingTime(MT1[rnd.randrange(0, len(MT1))])  
 newClass.set\_meetingTime1(dbMgr.get\_meetingTimes()[0])  
 elif (subjects[j].get\_numbHour() == 2):  
 call\_funcs\_randomly([wholetwohours, splittwohours])  
 elif (subjects[j].get\_numbHour() == 3):  
 call\_funcs\_randomly([wholethreehours, splitthreehours])  
 elif (subjects[j].get\_numbHour() == 4):  
 call\_funcs\_randomly([wholefourhours, splitfourhours])  
 elif (subjects[j].get\_numbHour() == 6):  
 newClass.set\_meetingTime(MT3[rnd.randrange(0, len(MT3))])  
 newClass.set\_meetingTime1(MT3v2[rnd.randrange(0, len(MT3v2))])  
 else:  
 newClass.set\_meetingTime(  
 dbMgr.get\_meetingTimes()[rnd.randrange(0, len(dbMgr.get\_meetingTimes()))])  
 newClass.set\_meetingTime1(  
 dbMgr.get\_meetingTimes()[rnd.randrange(0, len(dbMgr.get\_meetingTimes()))])  
 else:  
 if (subjects[j].get\_numbHour() == 1):  
 newClass.set\_meetingTime(MT1[rnd.randrange(0, len(MT1))])  
 newClass.set\_meetingTime1(dbMgr.get\_meetingTimes()[0])  
 elif (subjects[j].get\_numbHour() == 2):  
 call\_funcs\_randomly([wholetwohours, splittwohours])  
 elif (subjects[j].get\_numbHour() == 3):  
 call\_funcs\_randomly([wholethreehours, splitthreehours, splittwoplusone])  
 elif (subjects[j].get\_numbHour() == 4):  
 call\_funcs\_randomly([wholefourhours, splitfourhours, splitthreeplusone])  
 elif (subjects[j].get\_numbHour() == 6):  
 call\_funcs\_randomly([splitsixhours, splitfourplustwo])  
 else:  
 newClass.set\_meetingTime(  
 dbMgr.get\_meetingTimes()[rnd.randrange(0, len(dbMgr.get\_meetingTimes()))])  
 newClass.set\_meetingTime1(  
 dbMgr.get\_meetingTimes()[rnd.randrange(0, len(dbMgr.get\_meetingTimes()))])  
 self.\_classes.append(newClass) *# Add result as new class* return self  
  
 def calculate\_fitness(self):  
 self.\_conflicts=[]  
 classes=self.get\_classes()  
 instructorNames=list()  
 instructor\_Hours=list()  
 for i in range(0, len(classes)):  
 instructorNames.append(classes[i].get\_instructor().get\_name())  
 subjectHours=list()  
 if (classes[i].get\_meetingTime1().get\_id() != **'NULL'**):  
 if (classes[i].get\_meetingTime().get\_id() == classes[i].get\_meetingTime1().get\_id()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(Conflict(Conflict.ConflictType.SAME\_MTS, conflictBetweenClasses))  
 if DISABLE\_UNEQUAL\_SPLIT == True:  
 if (classes[i].get\_meetingTime().get\_MThour() != classes[  
 i].get\_meetingTime1().get\_MThour()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.UNEQUAL\_SPLIT, conflictBetweenClasses))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 i].get\_meetingTime1().get\_restTime()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.OVERLAP\_MTS, conflictBetweenClasses))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 i].get\_meetingTime().get\_restTime()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.OVERLAP\_MTS, conflictBetweenClasses))  
 if DISABLE\_CASUAL\_SPLITTING == True:  
 *# Disable Same Day Split Time* if (classes[i].get\_meetingTime().get\_day() == classes[i].get\_meetingTime1().get\_day()):  
 if (classes[i].get\_meetingTime().get\_MThour() != classes[  
 i].get\_subject().get\_numbHour()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.CASUAL\_SPLITTING, conflictBetweenClasses))  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[i].get\_meetingTime1().get\_MThour()) !=  
 classes[i].get\_subject().get\_numbHour()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.WRONG\_MTHOUR, conflictBetweenClasses))  
 if ENABLE\_NUMB\_OF\_STUDENTS == True:  
 if (classes[i].get\_room().get\_seatingCapacity() < classes[  
 i].get\_subject().get\_maxNumbOfStudents()):  
 seatingCapacityConflict=list()  
 seatingCapacityConflict.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.NUMB\_OF\_STUDENTS, seatingCapacityConflict))  
 *# Disable to assign Labs on VR* if LAB\_ON\_VR == False:  
 if (classes[i].get\_subject().get\_compatibility() == **'PR'**):  
 if (classes[i].get\_room().get\_type() != **'PR'**):  
 roomCompatibilityConflict=list()  
 roomCompatibilityConflict.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.LAB\_ON\_VR, roomCompatibilityConflict))  
 *# Disable cross department room utility* if DISABLE\_XDEPT\_ROOM\_UTILITY == True:  
 if (classes[i].get\_room().get\_type() != **'VR'**):  
 if (classes[i].get\_subject().get\_dept() != classes[i].get\_room().get\_dept()):  
 roomUtilityConflict=list()  
 roomUtilityConflict.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.XDEPT\_ROOM\_UTILITY, roomUtilityConflict))  
  
 *# if ENABLE\_INSTRUCTOR\_OVERTIME == True:* if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 0* if (classes[i].get\_meetingTime().get\_id() in classes[  
 i].get\_instructor().get\_overtime()): *# This identify the schedules inside the overtime* if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_OVERTIME, conflictBetweenClasses))  
 else:  
 *# 0* if (classes[i].get\_meetingTime().get\_id() in classes[i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_instructor().get\_availability()): *# This removes the schedules out side the official time and overtime* conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_OVERTIME, conflictBetweenClasses))  
 *# 1* if (classes[i].get\_meetingTime1().get\_id() in classes[i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime1().get\_id() not in classes[  
 i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_OVERTIME, conflictBetweenClasses))  
 *# Limit Instructor Availibilty* if LIMIT\_INSTRUCTOR\_AVAILABILITY == True:  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_AVAILABILITY,  
 conflictBetweenClasses))  
 else:  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_AVAILABILITY,  
 conflictBetweenClasses))  
 *# 1* if (classes[i].get\_meetingTime1().get\_id() not in classes[  
 i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime1().get\_id() not in classes[  
 i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_AVAILABILITY,  
 conflictBetweenClasses))  
 *# Limit Room Availibilty* if LIMIT\_ROOM\_AVAILABILITY == True:  
 if (classes[i].get\_room().get\_type() != **'VR'**):  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_room().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_AVAILABILITY, conflictBetweenClasses))  
 else:  
 *# 1* if (classes[i].get\_meetingTime1().get\_id() not in classes[  
 i].get\_room().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_AVAILABILITY, conflictBetweenClasses))  
 *# Limit Section Availibilty* if LIMIT\_SECTION\_AVAILABILITY == True:  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_section().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_AVAILABILITY, conflictBetweenClasses))  
 else:  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_section().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_AVAILABILITY, conflictBetweenClasses))  
 *# 1* if (classes[i].get\_meetingTime1().get\_id() not in classes[  
 i].get\_section().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_AVAILABILITY, conflictBetweenClasses))  
  
 for j in range(0, len(classes)):  
 *# Class Optimization* if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day() and  
 classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[  
 j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 else:  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day() and  
 classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[  
 j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 *# 11* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime1().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime1().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[  
 j].get\_meetingTime1().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime1().get\_breakTime()):  
 if ((classes[i].get\_meetingTime1().get\_MThour() + classes[  
 j].get\_meetingTime1().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 *# 10* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime1().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime1().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime1().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime1().get\_MThour() + classes[  
 j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 *# 01* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime1().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[  
 j].get\_meetingTime1().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 *# Instructor Optimization* if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[  
 j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 else:  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[  
 j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# 11* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime1().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime1().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[  
 j].get\_meetingTime1().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime1().get\_breakTime()):  
 if ((classes[i].get\_meetingTime1().get\_MThour() + classes[  
 j].get\_meetingTime1().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# 01* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime1().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime1().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime1().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime1().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[  
 j].get\_meetingTime1().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# 10* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime1().get\_MThour() + classes[  
 j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# Room Optimization* if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day() and  
 classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 else:  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day() and  
 classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 *# 11* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime1().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime1().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[  
 j].get\_meetingTime1().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 *# 01* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime1().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime1().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime1().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 *# 10* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 if (j >= i):  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_meetingTime().get\_id() == classes[j].get\_meetingTime().get\_id()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, roomBookingConflict))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 else:  
 *# 00* if (classes[i].get\_meetingTime().get\_id() == classes[j].get\_meetingTime().get\_id()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, roomBookingConflict))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# 11* if (classes[i].get\_meetingTime1().get\_id() == classes[j].get\_meetingTime1().get\_id()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, roomBookingConflict))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# 01* if (classes[i].get\_meetingTime().get\_id() == classes[j].get\_meetingTime1().get\_id()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, roomBookingConflict))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# 10* if (classes[i].get\_meetingTime1().get\_id() == classes[j].get\_meetingTime().get\_id()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, roomBookingConflict))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# Disable Dual Class Type* if DISABLE\_MIXED\_TYPE == True:  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime().get\_day() == classes[  
 j].get\_meetingTime().get\_day()):   
 if (classes[i].get\_room().get\_type() != classes[  
 j].get\_room().get\_type()):   
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 roomBookingConflict)) *# Room Type Conflict* else:  
 *# 00* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime().get\_day() == classes[  
 j].get\_meetingTime().get\_day()):   
 if (classes[i].get\_room().get\_type() != classes[  
 j].get\_room().get\_type()):   
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 roomBookingConflict)) *# Room Type Conflict  
 # 11* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime1().get\_day() == classes[  
 j].get\_meetingTime1().get\_day()):   
 if (classes[i].get\_room().get\_type() != classes[  
 j].get\_room().get\_type()):   
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 roomBookingConflict)) *# Room Type Conflict  
 # 01* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime().get\_day() == classes[  
 j].get\_meetingTime1().get\_day()):   
 if (classes[i].get\_room().get\_type() != classes[  
 j].get\_room().get\_type()):   
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 roomBookingConflict)) *# Room Type Conflict  
 # 10* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime1().get\_day() == classes[  
 j].get\_meetingTime().get\_day()):   
 if (classes[i].get\_room().get\_type() != classes[  
 j].get\_room().get\_type()):   
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 roomBookingConflict)) *# Room Type Conflict* if (classes[j].get\_instructor().get\_name() == instructorNames[i]):  
 subjectHours.append(float(classes[j].get\_subject().get\_numbHour()))  
 *# Enable Unit Limit* instructor\_Hours.append(subjectHours)  
 if DISABLE\_INSTRUCTOR\_OVERLOAD == True:  
 if (classes[i].get\_instructor().get\_max\_hours() < sum(instructor\_Hours[i])):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_OVERLOAD, instructorBookingConflict))  
 return 1 / ((1.0 \* len(self.\_conflicts) + 1))  
  
 def \_\_str\_\_(self):  
 returnValue=**""** for i in range(0, len(self.\_classes) - 1):  
 returnValue+=str(self.\_classes[i]) + **", "** returnValue+=str(self.\_classes[len(self.\_classes) - 1])  
 return returnValue  
  
 class Population:  
 def \_\_init\_\_(self, size):  
 self.\_size=size  
 self.\_data=dbMgr  
 self.\_schedules=[]  
 for i in range(0, size): self.\_schedules.append(Schedule().initialize())  
  
 def get\_size(self): return self.\_size  
  
 def get\_schedules(self): return self.\_schedules  
  
 class GeneticAlgorithm:  
 def evolve(self, population):  
 return self.\_mutate\_population(self.\_crossover\_population(population))  
  
 def \_crossover\_population(self, pop):  
 crossover\_pop=Population(0)  
 for i in range(NUMB\_OF\_ELITE\_SCHEDULES):  
 crossover\_pop.get\_schedules().append(pop.get\_schedules()[i])  
 i=NUMB\_OF\_ELITE\_SCHEDULES  
 while i < POPULATION\_SIZE:  
 schedule1=self.\_select\_tournament\_population(pop).get\_schedules()[0]  
 schedule2=self.\_select\_tournament\_population(pop).get\_schedules()[0]  
 crossover\_pop.get\_schedules().append(self.\_crossover\_schedule(schedule1, schedule2))  
 i+=1  
 return crossover\_pop  
  
 def \_mutate\_population(self, population):  
 for i in range(NUMB\_OF\_ELITE\_SCHEDULES, POPULATION\_SIZE):  
 self.\_mutate\_schedule(population.get\_schedules()[i])  
 return population  
  
 def \_crossover\_schedule(self, schedule1, schedule2):  
 crossoverSchedule=Schedule().initialize()  
 for i in range(0, len(crossoverSchedule.get\_classes())):  
 if (rnd.random() > 0.5):  
 crossoverSchedule.get\_classes()[i]=schedule1.get\_classes()[i]  
 else:  
 crossoverSchedule.get\_classes()[i]=schedule2.get\_classes()[i]  
 return crossoverSchedule  
  
 def \_mutate\_schedule(self, mutateSchedule):  
 schedule=Schedule().initialize()  
 for i in range(0, len(mutateSchedule.get\_classes())):  
 if (MUTATION\_RATE > rnd.random()): mutateSchedule.get\_classes()[i]=schedule.get\_classes()[i]  
 return mutateSchedule  
  
 def \_select\_tournament\_population(self, pop):  
 tournament\_pop=Population(0)  
 i=0  
 while i < TOURNAMENT\_SELECTION\_SIZE:  
 tournament\_pop.get\_schedules().append(  
 pop.get\_schedules()[rnd.randrange(0, int(POPULATION\_SIZE))])  
 i+=1  
 tournament\_pop.get\_schedules().sort(key = lambda x: x.get\_fitness(), reverse = True)  
 return tournament\_pop  
  
 class Section:  
 def \_\_init\_\_(self, code, subjects, availability):  
 self.\_code=code  
 self.\_subjects=subjects  
 self.\_availability=availability  
  
 def get\_code(self): return self.\_code  
  
 def get\_subjects(self): return self.\_subjects  
  
 def get\_availability(self): return self.\_availability  
  
 class Subject:  
 def \_\_init\_\_(self, code, description, units, numbHour, compatibility, curriculum, maxNumbOfStudents,  
 dept,  
 instructors):  
 self.\_code=code  
 self.\_description=description  
 self.\_units=units  
 self.\_numbHour=numbHour  
 self.\_compatibility=compatibility  
 self.\_curriculum=curriculum  
 self.\_maxNumbOfStudents=maxNumbOfStudents  
 self.\_dept=dept  
 self.\_instructors=instructors  
  
 def get\_code(self): return self.\_code  
  
 def get\_description(self): return self.\_description  
  
 def get\_units(self): return self.\_units  
  
 def get\_numbHour(self): return self.\_numbHour  
  
 def get\_compatibility(self): return self.\_compatibility  
  
 def get\_curriculum(self): return self.\_curriculum  
  
 def get\_maxNumbOfStudents(self): return self.\_maxNumbOfStudents  
  
 def get\_dept(self): return self.\_dept  
  
 def get\_instructors(self): return self.\_instructors  
  
 def \_\_str\_\_(self): return self.\_description  
  
 class Time:  
 def \_\_init\_\_(self, block):  
 self.\_block=block  
  
 def get\_block(self): return self.\_block  
  
 def \_\_str\_\_(self): return self.\_block  
  
 class TimePoint:  
 def \_\_init\_\_(self, point, blocks, phantomBlocks):  
 self.\_point=point  
 self.\_blocks=blocks  
 self.\_phantomBlocks=phantomBlocks  
  
 def get\_point(self): return self.\_point  
  
 def get\_blocks(self): return self.\_blocks  
  
 def get\_phantomBlocks(self): return self.\_phantomBlocks  
  
 def \_\_str\_\_(self): return self.\_point  
  
 class Instructor:  
 def \_\_init\_\_(self, number, name, max\_hours, seniority, availability, start, end, overtime):  
 self.\_number=number  
 self.\_name=name  
 self.\_max\_hours=max\_hours  
 self.\_seniority=seniority  
 self.\_availability=availability  
 self.\_start=start  
 self.\_end=end  
 self.\_overtime=overtime  
  
 def get\_number(self): return self.\_number  
  
 def get\_name(self): return self.\_name  
  
 def get\_max\_hours(self): return self.\_max\_hours  
  
 def get\_seniority(self): return self.\_seniority  
  
 def get\_availability(self): return self.\_availability  
  
 def get\_start(self): return self.\_start  
  
 def get\_end(self): return self.\_end  
  
 def get\_overtime(self): return self.\_overtime  
  
 def \_\_str\_\_(self): return self.\_name  
  
 class MeetingTime:  
 def \_\_init\_\_(self, id, time, day, MThour, cell, restTime, breakTime):   
 self.\_id=id  
 self.\_time=time  
 self.\_day=day   
 self.\_MThour=MThour  
 self.\_cell=cell  
 self.\_restTime=restTime   
 self.\_breakTime=breakTime  
  
 def get\_id(self): return self.\_id  
  
 def get\_time(self): return self.\_time  
  
 def get\_day(self): return self.\_day   
  
 def get\_MThour(self): return self.\_MThour  
  
 def get\_cell(self): return self.\_cell  
  
 def get\_restTime(self): return self.\_restTime   
  
 def get\_breakTime(self): return self.\_breakTime  
  
 def \_\_str\_\_(self): return self.\_id  
  
 class MeetingTime1:  
 def \_\_init\_\_(self, id, time, day, MThour, cell, restTime, breakTime):   
 self.\_id=id  
 self.\_time=time  
 self.\_day=day   
 self.\_MThour=MThour  
 self.\_cell=cell  
 self.\_restTime=restTime   
 self.\_breakTime=breakTime  
  
 def get\_id(self): return self.\_id  
  
 def get\_time(self): return self.\_time  
  
 def get\_day(self): return self.\_day   
  
 def get\_MThour(self): return self.\_MThour  
  
 def get\_cell(self): return self.\_cell  
  
 def get\_restTime(self): return self.\_restTime   
  
 def get\_breakTime(self): return self.\_breakTime  
  
 def \_\_str\_\_(self): return self.\_id  
  
 class Room:  
 def \_\_init\_\_(self, number, seatingCapacity, type, dept, availability):   
 self.\_number=number  
 self.\_seatingCapacity=seatingCapacity  
 self.\_type=type   
 self.\_dept=dept  
 self.\_availability=availability  
  
 def get\_number(self): return self.\_number  
  
 def get\_seatingCapacity(self): return self.\_seatingCapacity  
  
 def get\_type(self): return self.\_type   
  
 def get\_dept(self): return self.\_dept  
  
 def get\_availability(self): return self.\_availability  
  
 def \_\_str\_\_(self): return self.\_number  
  
 class Class:  
 def \_\_init\_\_(self, id, section, subject):  
 self.\_id=id  
 self.\_section=section  
 self.\_subject=subject  
 self.\_instructor=None  
 self.\_meetingTime=None  
 self.\_meetingTime1=None  
 self.\_room=None  
  
 def get\_id(self): return self.\_id  
  
 def get\_section(self): return self.\_section  
  
 def get\_subject(self): return self.\_subject  
  
 def get\_instructor(self): return self.\_instructor  
  
 def get\_meetingTime(self): return self.\_meetingTime  
  
 def get\_meetingTime1(self): return self.\_meetingTime1  
  
 def get\_room(self): return self.\_room  
  
 def set\_instructor(self, instructor): self.\_instructor=instructor  
  
 def set\_meetingTime(self, meetingTime): self.\_meetingTime=meetingTime  
  
 def set\_meetingTime1(self, meetingTime1): self.\_meetingTime1=meetingTime1  
  
 def set\_room(self, room): self.\_room=room  
  
 def \_\_str\_\_(self):  
 return str(self.\_section.get\_code()) + **","** + str(self.\_subject.get\_code()) + **","** + \  
 str(self.\_room.get\_number()) + **","** + str(self.\_instructor.get\_number()) + **","** + str(  
 self.\_meetingTime.get\_id()) + **","** + str(self.\_meetingTime1.get\_id())  
  
 class Conflict:  
 class ConflictType(Enum):  
 WRONG\_MTHOUR=1  
 UNEQUAL\_SPLIT=2  
 CASUAL\_SPLITTING=3  
 SAME\_MTS=4  
 OVERLAP\_MTS=5  
  
 NUMB\_OF\_STUDENTS=6  
 ROOM\_AVAILABILITY=7  
 ROOM\_BOOKING=8  
 XDEPT\_ROOM\_UTILITY=9  
  
 INSTRUCTOR\_OVERTIME=10  
 INSTRUCTOR\_AVAILABILITY=11  
 INSTRUCTOR\_BOOKING=12  
 INSTRUCTOR\_OVERLOAD=13  
  
 SECTION\_AVAILABILITY=14  
 SECTION\_BOOKING=15  
  
 LAB\_ON\_VR=16   
 MIXED\_TYPE=17  
  
 def \_\_init\_\_(self, conflictType, conflictBetweenClasses):  
 self.\_conflictType=conflictType  
 self.\_conflictBetweenClasses=conflictBetweenClasses  
  
 def get\_conflictType(self): return self.\_conflictType  
  
 def get\_conflictBetweenClasses(self): return self.\_conflictBetweenClasses  
  
 def \_\_str\_\_(self): return str(self.\_conflictType) + **" "** + str(  
 **" and "**.join(map(str, self.\_conflictBetweenClasses)))  
  
 def click(event):  
 try:  
 selected=tree.focus() *# Grab record position/number* values=tree.item(selected, **'values'**) *# Grab record values* cb\_room\_number.delete(0, **'end'**)  
 cb\_meeting\_time.delete(0, **'end'**)  
 cb\_room\_number.insert(0, values[0])  
 cb\_meeting\_time.insert(0, values[1])  
 except IndexError: pass  
  
 def combobox\_input0():  
 ARADB()  
 cursor.execute(**"SELECT number FROM `room`"**)  
 conn.commit()  
 data=[]  
 for row in cursor.fetchall():  
 data.append(row[0])  
 return data  
 cursor.close()  
 conn.close()  
  
 def combobox\_input1():  
 ARADB()  
 cursor.execute(**"SELECT id FROM `meeting\_time`"**)  
 conn.commit()  
 data=[]  
 for row in cursor.fetchall():  
 data.append(row[0])  
 return data  
 cursor.close()  
 conn.close()  
  
 def combobox\_input2():  
 ARADB()  
 cursor.execute(**"SELECT point FROM `time\_point`"**)  
 conn.commit()  
 data=[]  
 for row in cursor.fetchall():  
 data.append(row[0])  
 return data  
 cursor.close()  
 conn.close()  
  
 def Create(): *# MT INPUTS* if ROOM\_NUMBER.get() == **""**:  
 txt\_result.config(text = **"Please choose a room!"**, fg = **"red"**)  
 else:  
 dbMgr=DBMgr()  
 meetingTimes=dbMgr.get\_meetingTimes()  
 timepoints=dbMgr.get\_timepoints()  
 sep=**' '** if (not (MON\_START.get() == **""**)):  
 ID\_Mon=list() *# Meeting Time IDs in Monday* Time\_Mon=list()  
 startingTime\_Mon=list()  
 endingTime\_Mon=list()  
 phantom\_time=list()  
 qualifiedMTs\_Mon=list()  
 qualifiedBlocks\_Mon=list()  
 *# List all Mondays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Monday'** == meetingTimes[i].get\_day()):  
 ID\_Mon.append(meetingTimes[i].get\_id())  
 Time\_Mon.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (MON\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Mon.append(timepoints[i].get\_blocks())  
 if (MON\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Mon.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (MON\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Mon)):  
 if (Time\_Mon[i] in startingTime\_Mon[0]):  
 if (Time\_Mon[i] not in endingTime\_Mon[0]):  
 if (Time\_Mon[i] not in phantom\_time[0]):  
 qualifiedMTs\_Mon.append(ID\_Mon[i])  
 qualifiedBlocks\_Mon.append(Time\_Mon[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Mon)):  
 if (Time\_Mon[i] in qualifiedBlocks\_Mon):  
 cursor.execute(  
 **"INSERT INTO 'room\_availability' (room\_number, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(ROOM\_NUMBER.get().split(sep, 1)[0]), str(ID\_Mon[i]), str(MON\_START.get()),  
 str(MON\_END.get())))  
 conn.commit()  
 MON\_START.set(**""**)  
 MON\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 if (not (TUE\_START.get() == **""**)):  
 ID\_Tue=list() *# Meeting Time IDs in Tuesday* Time\_Tue=list()  
 startingTime\_Tue=list()  
 endingTime\_Tue=list()  
 phantom\_time=list()  
 qualifiedMTs\_Tue=list()  
 qualifiedBlocks\_Tue=list()  
 *# List all Tuesdays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Tuesday'** == meetingTimes[i].get\_day()):  
 ID\_Tue.append(meetingTimes[i].get\_id())  
 Time\_Tue.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (TUE\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Tue.append(timepoints[i].get\_blocks())  
 if (TUE\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Tue.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (TUE\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Tue)):  
 if (Time\_Tue[i] in startingTime\_Tue[0]):  
 if (Time\_Tue[i] not in endingTime\_Tue[0]):  
 if (Time\_Tue[i] not in phantom\_time[0]):  
 qualifiedMTs\_Tue.append(ID\_Tue[i])  
 qualifiedBlocks\_Tue.append(Time\_Tue[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Tue)):  
 if (Time\_Tue[i] in qualifiedBlocks\_Tue):  
 cursor.execute(  
 **"INSERT INTO 'room\_availability' (room\_number, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(ROOM\_NUMBER.get().split(sep, 1)[0]), str(ID\_Tue[i]), str(TUE\_START.get()),  
 str(TUE\_END.get())))  
 conn.commit()  
 TUE\_START.set(**""**)  
 TUE\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 if (not (WED\_START.get() == **""**)):  
 ID\_Wed=list() *# Meeting Time IDs in Wednesday* Time\_Wed=list()  
 startingTime\_Wed=list()  
 endingTime\_Wed=list()  
 phantom\_time=list()  
 qualifiedMTs\_Wed=list()  
 qualifiedBlocks\_Wed=list()  
 *# List all Wednesdays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Wednesday'** == meetingTimes[i].get\_day()):  
 ID\_Wed.append(meetingTimes[i].get\_id())  
 Time\_Wed.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (WED\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Wed.append(timepoints[i].get\_blocks())  
 if (WED\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Wed.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (WED\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Wed)):  
 if (Time\_Wed[i] in startingTime\_Wed[0]):  
 if (Time\_Wed[i] not in endingTime\_Wed[0]):  
 if (Time\_Wed[i] not in phantom\_time[0]):  
 qualifiedMTs\_Wed.append(ID\_Wed[i])  
 qualifiedBlocks\_Wed.append(Time\_Wed[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Wed)):  
 if (Time\_Wed[i] in qualifiedBlocks\_Wed):  
 cursor.execute(  
 **"INSERT INTO 'room\_availability' (room\_number, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(ROOM\_NUMBER.get().split(sep, 1)[0]), str(ID\_Wed[i]), str(WED\_START.get()),  
 str(WED\_END.get())))  
 conn.commit()  
 WED\_START.set(**""**)  
 WED\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 if (not (THU\_START.get() == **""**)):  
 ID\_Thu=list() *# Meeting Time IDs in Thursday* Time\_Thu=list()  
 startingTime\_Thu=list()  
 endingTime\_Thu=list()  
 phantom\_time=list()  
 qualifiedMTs\_Thu=list()  
 qualifiedBlocks\_Thu=list()  
 *# List all Thursdays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Thursday'** == meetingTimes[i].get\_day()):  
 ID\_Thu.append(meetingTimes[i].get\_id())  
 Time\_Thu.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (THU\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Thu.append(timepoints[i].get\_blocks())  
 if (THU\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Thu.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (THU\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Thu)):  
 if (Time\_Thu[i] in startingTime\_Thu[0]):  
 if (Time\_Thu[i] not in endingTime\_Thu[0]):  
 if (Time\_Thu[i] not in phantom\_time[0]):  
 qualifiedMTs\_Thu.append(ID\_Thu[i])  
 qualifiedBlocks\_Thu.append(Time\_Thu[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Thu)):  
 if (Time\_Thu[i] in qualifiedBlocks\_Thu):  
 cursor.execute(  
 **"INSERT INTO 'room\_availability' (room\_number, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(ROOM\_NUMBER.get().split(sep, 1)[0]), str(ID\_Thu[i]), str(THU\_START.get()),  
 str(THU\_END.get())))  
 conn.commit()  
 THU\_START.set(**""**)  
 THU\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 if (not (FRI\_START.get() == **""**)):  
 ID\_Fri=list() *# Meeting Time IDs in Friday* Time\_Fri=list()  
 startingTime\_Fri=list()  
 endingTime\_Fri=list()  
 phantom\_time=list()  
 qualifiedMTs\_Fri=list()  
 qualifiedBlocks\_Fri=list()  
 *# List all Fridays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Friday'** == meetingTimes[i].get\_day()):  
 ID\_Fri.append(meetingTimes[i].get\_id())  
 Time\_Fri.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (FRI\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Fri.append(timepoints[i].get\_blocks())  
 if (FRI\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Fri.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (FRI\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Fri)):  
 if (Time\_Fri[i] in startingTime\_Fri[0]):  
 if (Time\_Fri[i] not in endingTime\_Fri[0]):  
 if (Time\_Fri[i] not in phantom\_time[0]):  
 qualifiedMTs\_Fri.append(ID\_Fri[i])  
 qualifiedBlocks\_Fri.append(Time\_Fri[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Fri)):  
 if (Time\_Fri[i] in qualifiedBlocks\_Fri):  
 cursor.execute(  
 **"INSERT INTO 'room\_availability' (room\_number, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(ROOM\_NUMBER.get().split(sep, 1)[0]), str(ID\_Fri[i]), str(FRI\_START.get()),  
 str(FRI\_END.get())))  
 conn.commit()  
 FRI\_START.set(**""**)  
 FRI\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 if (not (SAT\_START.get() == **""**)):  
 ID\_Sat=list() *# Meeting Time IDs in Saturday* Time\_Sat=list()  
 startingTime\_Sat=list()  
 endingTime\_Sat=list()  
 phantom\_time=list()  
 qualifiedMTs\_Sat=list()  
 qualifiedBlocks\_Sat=list()  
 *# List all Saturdays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Saturday'** == meetingTimes[i].get\_day()):  
 ID\_Sat.append(meetingTimes[i].get\_id())  
 Time\_Sat.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (SAT\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Sat.append(timepoints[i].get\_blocks())  
 if (SAT\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Sat.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (SAT\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Sat)):  
 if (Time\_Sat[i] in startingTime\_Sat[0]):  
 if (Time\_Sat[i] not in endingTime\_Sat[0]):  
 if (Time\_Sat[i] not in phantom\_time[0]):  
 qualifiedMTs\_Sat.append(ID\_Sat[i])  
 qualifiedBlocks\_Sat.append(Time\_Sat[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Sat)):  
 if (Time\_Sat[i] in qualifiedBlocks\_Sat):  
 cursor.execute(  
 **"INSERT INTO 'room\_availability' (room\_number, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(ROOM\_NUMBER.get().split(sep, 1)[0]), str(ID\_Sat[i]), str(SAT\_START.get()),  
 str(SAT\_END.get())))  
 conn.commit()  
 SAT\_START.set(**""**)  
 SAT\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 if (not (SUN\_START.get() == **""**)):  
 ID\_Sun=list() *# Meeting Time IDs in Sunday* Time\_Sun=list()  
 startingTime\_Sun=list()  
 endingTime\_Sun=list()  
 phantom\_time=list()  
 qualifiedMTs\_Sun=list()  
 qualifiedBlocks\_Sun=list()  
 *# List all Sundays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Sunday'** == meetingTimes[i].get\_day()):  
 ID\_Sun.append(meetingTimes[i].get\_id())  
 Time\_Sun.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (SUN\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Sun.append(timepoints[i].get\_blocks())  
 if (SUN\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Sun.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (SUN\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Sun)):  
 if (Time\_Sun[i] in startingTime\_Sun[0]):  
 if (Time\_Sun[i] not in endingTime\_Sun[0]):  
 if (Time\_Sun[i] not in phantom\_time[0]):  
 qualifiedMTs\_Sun.append(ID\_Sun[i])  
 qualifiedBlocks\_Sun.append(Time\_Sun[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Sun)):  
 if (Time\_Sun[i] in qualifiedBlocks\_Sun):  
 cursor.execute(  
 **"INSERT INTO 'room\_availability' (room\_number, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(ROOM\_NUMBER.get().split(sep, 1)[0]), str(ID\_Sun[i]), str(SUN\_START.get()),  
 str(SUN\_END.get())))  
 conn.commit()  
 SUN\_START.set(**""**)  
 SUN\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Created a data!"**, fg = **"green"**)  
  
 def View():  
 tree.delete(\*tree.get\_children())  
 ARADB()  
 cursor.execute(**"SELECT \* FROM `room\_availability` ORDER BY `room\_number` ASC"**)  
 fetch=cursor.fetchall()  
 for data in fetch:  
 tree.insert(**''**, **'end'**, values = (data[0], data[1], data[2], data[3]))  
 cursor.close()  
 conn.close()  
 txt\_result.config(text = **"Successfully viewed the data from database"**, fg = **"black"**)  
  
 def Delete():  
 if ROOM\_NUMBER.get() == **""**:  
 txt\_result.config(text = **"Please choose a room!"**, fg = **"red"**)  
 else:  
 ARADB()  
 cursor.execute(  
 **"DELETE FROM 'room\_availability' WHERE room\_number = '"** + str(ROOM\_NUMBER.get()) + **"'"**)  
 conn.commit()  
 cb\_room\_number.delete(0, **'end'**)  
 cb\_meeting\_time.delete(0, **'end'**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Deleted Successfully!"**, fg = **"green"**)  
  
 def Clear():  
 result=messagebox.askquestion(**"Clear"**, **"This will clear the Room Availability table.**\n**Do you want to proceed?"**, icon = **'warning'**)  
 if result == **'yes'**:  
 ARADB()  
 cursor.execute(**"""DROP TABLE room\_availability"""**)  
 cursor.execute(**"""create table room\_availability (room\_number text NOT NULL, meeting\_time\_id text NOT NULL, start text, end text)"""**)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Table Cleared Successfully!"**, fg = **"green"**)  
 else:  
 pass  
  
 def Refresh():  
 RA.destroy()  
 Modify\_Availability()  
  
 *# ==================================VARIABLES==========================================* ROOM\_NUMBER=StringVar(RA)  
 MEETING\_TIME=StringVar(RA)  
 MON\_START=StringVar(RA)  
 MON\_END=StringVar(RA)  
 TUE\_START=StringVar(RA)  
 TUE\_END=StringVar(RA)  
 WED\_START=StringVar(RA)  
 WED\_END=StringVar(RA)  
 THU\_START=StringVar(RA)  
 THU\_END=StringVar(RA)  
 FRI\_START=StringVar(RA)  
 FRI\_END=StringVar(RA)  
 SAT\_START=StringVar(RA)  
 SAT\_END=StringVar(RA)  
 SUN\_START=StringVar(RA)  
 SUN\_END=StringVar(RA)  
  
 *# ==================================FRAME==============================================* Top=Frame(RA, width = 900, height = 50, bd = 8, relief = **"raise"**, bg = **"#FEB144"**)  
 Top.pack(side = TOP)  
 Left=Frame(RA, width = 300, height = 500, bd = 8, relief = **"raise"**)  
 Left.pack(side = LEFT)  
 Right=Frame(RA, width = 600, height = 500, bd = 8, relief = **"raise"**)  
 Right.pack(side = RIGHT)  
 Forms=Frame(Left, width = 300, height = 450)  
 Forms.pack(side = TOP)  
 Buttons=Frame(Left, width = 300, height = 100, bd = 8, relief = **"raise"**)  
 Buttons.pack(side = BOTTOM)  
  
 *# ==================================LABEL WIDGET=======================================* txt\_title=Label(Top, width = 900, font = (**'arial'**, 24), text = **"Room Availability"**, fg = **"Black"**,  
 bg = **"#ffd587"**)   
 txt\_title.pack()  
 txt\_instructor=Label(Forms, text = **"Select Room:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_instructor.grid(row = 0, stick = **"e"**)  
  
 txt\_monday=Label(Forms, text = **"Monday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_monday.grid(row = 1, stick = **"e"**)  
 txt\_tuesday=Label(Forms, text = **"Tuesday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_tuesday.grid(row = 2, stick = **"e"**)  
 txt\_wednesday=Label(Forms, text = **"Wednesday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_wednesday.grid(row = 3, stick = **"e"**)  
 txt\_thursday=Label(Forms, text = **"Thursday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_thursday.grid(row = 4, stick = **"e"**)  
 txt\_friday=Label(Forms, text = **"Friday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_friday.grid(row = 5, stick = **"e"**)  
 txt\_saturday=Label(Forms, text = **"Saturday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_saturday.grid(row = 6, stick = **"e"**)  
 txt\_sunday=Label(Forms, text = **"Sunday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_sunday.grid(row = 7, stick = **"e"**)  
 txt\_result=Label(Buttons)  
 txt\_result.pack(side = TOP)  
  
 *# ==================================ENTRY WIDGET=======================================* cb\_room\_number=ttk.Combobox(Forms, textvariable = ROOM\_NUMBER, width = 44)  
 cb\_room\_number[**'values'**]=combobox\_input0()  
 cb\_room\_number.place(x = 180, y = 18)  
 cb\_meeting\_time=ttk.Combobox(Forms, textvariable = MEETING\_TIME, width = 20)  
 cb\_meeting\_time[**'values'**]=combobox\_input1()  
  
 txt\_monstart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_monstart.grid(column = 1, row = 1, stick = **"w"**)  
 cb\_monstart=ttk.Combobox(Forms, textvariable = MON\_START, width = 10)  
 cb\_monstart[**'values'**]=combobox\_input2()  
 cb\_monstart.grid(column = 2, row = 1, stick = **"w"**)  
 txt\_monend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 12)  
 txt\_monend.grid(column = 3, row = 1, stick = **"w"**)  
 cb\_monend=ttk.Combobox(Forms, textvariable = MON\_END, width = 10)  
 cb\_monend[**'values'**]=combobox\_input2()  
 cb\_monend.grid(column = 4, row = 1, padx = 10)  
  
 txt\_tuestart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_tuestart.grid(column = 1, row = 2, stick = **"w"**)  
 cb\_tuestart=ttk.Combobox(Forms, textvariable = TUE\_START, width = 10)  
 cb\_tuestart[**'values'**]=combobox\_input2()  
 cb\_tuestart.grid(column = 2, row = 2, stick = **"w"**)  
 txt\_tueend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_tueend.grid(column = 3, row = 2, stick = **"w"**)  
 cb\_tueend=ttk.Combobox(Forms, textvariable = TUE\_END, width = 10)  
 cb\_tueend[**'values'**]=combobox\_input2()  
 cb\_tueend.grid(column = 4, row = 2, padx = 10)  
  
 txt\_wedstart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_wedstart.grid(column = 1, row = 3, stick = **"w"**)  
 cb\_wedstart=ttk.Combobox(Forms, textvariable = WED\_START, width = 10)  
 cb\_wedstart[**'values'**]=combobox\_input2()  
 cb\_wedstart.grid(column = 2, row = 3, stick = **"w"**)  
 txt\_wedend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_wedend.grid(column = 3, row = 3, stick = **"w"**)  
 cb\_wedend=ttk.Combobox(Forms, textvariable = WED\_END, width = 10)  
 cb\_wedend[**'values'**]=combobox\_input2()  
 cb\_wedend.grid(column = 4, row = 3, padx = 10)  
  
 txt\_thustart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_thustart.grid(column = 1, row = 4, stick = **"w"**)  
 cb\_thustart=ttk.Combobox(Forms, textvariable = THU\_START, width = 10)  
 cb\_thustart[**'values'**]=combobox\_input2()  
 cb\_thustart.grid(column = 2, row = 4, stick = **"w"**)  
 txt\_thuend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_thuend.grid(column = 3, row = 4, stick = **"w"**)  
 cb\_thuend=ttk.Combobox(Forms, textvariable = THU\_END, width = 10)  
 cb\_thuend[**'values'**]=combobox\_input2()  
 cb\_thuend.grid(column = 4, row = 4, padx = 10)  
  
 txt\_fristart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_fristart.grid(column = 1, row = 5, stick = **"w"**)  
 cb\_fristart=ttk.Combobox(Forms, textvariable = FRI\_START, width = 10)  
 cb\_fristart[**'values'**]=combobox\_input2()  
 cb\_fristart.grid(column = 2, row = 5, stick = **"w"**)  
 txt\_friend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_friend.grid(column = 3, row = 5, stick = **"w"**)  
 cb\_friend=ttk.Combobox(Forms, textvariable = FRI\_END, width = 10)  
 cb\_friend[**'values'**]=combobox\_input2()  
 cb\_friend.grid(column = 4, row = 5, padx = 10)  
  
 txt\_satstart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_satstart.grid(column = 1, row = 6, stick = **"w"**)  
 cb\_satstart=ttk.Combobox(Forms, textvariable = SAT\_START, width = 10)  
 cb\_satstart[**'values'**]=combobox\_input2()  
 cb\_satstart.grid(column = 2, row = 6, stick = **"w"**)  
 txt\_satend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_satend.grid(column = 3, row = 6, stick = **"w"**)  
 cb\_satend=ttk.Combobox(Forms, textvariable = SAT\_END, width = 10)  
 cb\_satend[**'values'**]=combobox\_input2()  
 cb\_satend.grid(column = 4, row = 6, padx = 10)  
  
 txt\_sunstart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_sunstart.grid(column = 1, row = 7, stick = **"w"**)  
 cb\_sunstart=ttk.Combobox(Forms, textvariable = SUN\_START, width = 10)  
 cb\_sunstart[**'values'**]=combobox\_input2()  
 cb\_sunstart.grid(column = 2, row = 7, stick = **"w"**)  
 txt\_sunend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_sunend.grid(column = 3, row = 7, stick = **"w"**)  
 cb\_sunend=ttk.Combobox(Forms, textvariable = SUN\_END, width = 10)  
 cb\_sunend[**'values'**]=combobox\_input2()  
 cb\_sunend.grid(column = 4, row = 7, padx = 10)  
  
 *# ==================================BUTTONS WIDGET=====================================* btn\_create=Button(Buttons, width = 10, text = **"Create"**, command = Create)  
 btn\_create.pack(side = LEFT)  
 btn\_view=Button(Buttons, width = 10, text = **"View"**, command = View)  
 btn\_view.pack(side = LEFT)  
 btn\_update=Button(Buttons, width = 10, text = **"Update"**, state = DISABLED)  
 btn\_update.pack(side = LEFT)  
 btn\_delete=Button(Buttons, width = 10, text = **"Delete"**, command = Delete)  
 btn\_delete.pack(side = LEFT)  
 btn\_clear=Button(Buttons, width = 10, text = **"Clear"**, fg = **"Black"**, bg = **"#F2B6AE"**, command = Clear)  
 btn\_clear.pack(side = LEFT)  
 btn\_refresh=Button(Top, width = 10, text = **"Refresh (F5)"**, font = (**'arial'**, 10, **'bold'**), fg = **"White"**,  
 bg = **"green"**, command = Refresh)  
 btn\_refresh.pack(side = RIGHT)  
  
 *# ==================================LIST WIDGET========================================* scrollbary=Scrollbar(Right, orient = VERTICAL)  
 scrollbarx=Scrollbar(Right, orient = HORIZONTAL)  
 columns=(**"Room"**, **"Meeting Time"**, **"Start"**, **"End"**)  
 tree=ttk.Treeview(Right, columns = columns, selectmode = **"extended"**, height = 500,  
 yscrollcommand = scrollbary.set, xscrollcommand = scrollbarx.set)  
  
 def treeview\_sort\_column(tree, col, reverse):  
 l=[(tree.set(k, col), k) for k in tree.get\_children(**''**)]  
 l.sort(reverse = reverse)  
  
 *# rearrange items in sorted positions* for index, (val, k) in enumerate(l):  
 tree.move(k, **''**, index)  
  
 *# reverse sort next time* tree.heading(col, command = lambda \_col=col: treeview\_sort\_column(tree, \_col, not reverse))  
  
 for col in columns:  
 tree.heading(col, text = col, command = lambda \_col=col: \  
 treeview\_sort\_column(tree, \_col, False))  
 *# for TREE Scroll Bars* scrollbary.config(command = tree.yview)  
 scrollbary.pack(side = RIGHT, fill = Y)  
 scrollbarx.config(command = tree.xview)  
 scrollbarx.pack(side = BOTTOM, fill = X)  
 *# inside the tree view* tree.heading(**'Room'**, text = **"Room"**, anchor = W)  
 tree.heading(**'Meeting Time'**, text = **"Meeting Time ID"**, anchor = W)  
 tree.heading(**'Start'**, text = **"Start"**, anchor = W)  
 tree.heading(**'End'**, text = **"End"**, anchor = W)  
 tree.column(**'#0'**, stretch = NO, minwidth = 0, width = 0)  
 tree.column(**'#1'**, stretch = NO, minwidth = 75, width = 75)  
 tree.column(**'#2'**, stretch = NO, minwidth = 160, width = 160)  
 tree.column(**'#3'**, stretch = NO, minwidth = 60, width = 60)  
 tree.bind(**'<ButtonRelease-1>'**, click)  
 tree.pack()  
  
 if \_\_name\_\_ == **'\_\_main\_\_'**:  
 View()  
 RA.mainloop()  
  
 def Refresh():  
 roomroot.destroy()  
 Room()  
  
 *# ==================================VARIABLES==========================================* NUMBER=StringVar(roomroot)  
 CAPACITY=IntVar(roomroot)  
 TYPE=StringVar(roomroot)  
 DEPT=StringVar(roomroot)  
  
 *# ==================================FRAME==============================================* Top=Frame(roomroot, width = 900, height = 50, bd = 8, relief = **"raise"**, bg = **"#FEB144"**)  
 Top.pack(side = TOP)  
 Left=Frame(roomroot, width = 300, height = 500, bd = 8, relief = **"raise"**)  
 Left.pack(side = LEFT)  
 Right=Frame(roomroot, width = 600, height = 500, bd = 8, relief = **"raise"**)  
 Right.pack(side = RIGHT)  
 Forms=Frame(Left, width = 300, height = 450)  
 Forms.pack(side = TOP)  
 Buttons=Frame(Left, width = 300, height = 100, bd = 8, relief = **"raise"**)  
 Buttons.pack(side = BOTTOM)  
 RadioGroup=Frame(Forms)  
 PR=Radiobutton(RadioGroup, text = **"Physical"**, variable = TYPE, value = **"PR"**, font = (**'arial'**, 16)).pack(side = LEFT)  
 VR=Radiobutton(RadioGroup, text = **"Virtual"**, variable = TYPE, value = **"VR"**, font = (**'arial'**, 16)).pack(side = LEFT)  
  
 *# ==================================LABEL WIDGET=======================================* txt\_title=Label(Top, width = 900, font = (**'arial'**, 24), text = **"Room"**, fg = **"Black"**,  
 bg = **"#FCDA9C"**)   
 txt\_title.pack()  
 txt\_number=Label(Forms, text = **"Room:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_number.grid(row = 0, stick = **"e"**)  
 txt\_capacity=Label(Forms, text = **"Capacity:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_capacity.grid(row = 1, stick = **"e"**)  
 txt\_type=Label(Forms, text = **"Type:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_type.grid(row = 2, stick = **"e"**)  
 txt\_result=Label(Buttons)  
 txt\_result.pack(side = TOP)  
 txt\_dept=Label(Forms, text = **"Select Dept:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_dept.grid(row = 3, stick = **"e"**)  
  
 *# ==================================ENTRY WIDGET=======================================* cb\_number=ttk.Combobox(Forms, textvariable = NUMBER, width = 20)  
 cb\_number[**'values'**]=combobox\_input0()  
 cb\_number.grid(column = 1, row = 0)  
 e\_capacity=Entry(Forms, textvariable = CAPACITY, width = 20)  
 e\_capacity.grid(row = 1, column = 1)  
 e\_type=Entry(Forms, textvariable = TYPE, width = 20)  
 RadioGroup.grid(row = 2, column = 1)  
 cb\_dept=ttk.Combobox(Forms, textvariable = DEPT, width = 20)  
 cb\_dept[**'values'**]=combobox\_input1()  
 cb\_dept.grid(column = 1, row = 3)  
  
 *# ==================================BUTTONS WIDGET=====================================* btn\_create=Button(Buttons, width = 10, text = **"Create"**, command = Create)  
 btn\_create.pack(side = LEFT)  
 btn\_view=Button(Buttons, width = 10, text = **"View"**, command = View)  
 btn\_view.pack(side = LEFT)  
 btn\_update=Button(Buttons, width = 10, text = **"Update"**, command = Update)  
 btn\_update.pack(side = LEFT)  
 btn\_delete=Button(Buttons, width = 10, text = **"Delete"**, command = Delete)  
 btn\_delete.pack(side = LEFT)  
 btn\_clear=Button(Buttons, width = 10, text = **"Clear"**, fg = **"Black"**, bg = **"#F2B6AE"**, command = Clear)  
 btn\_clear.pack(side = LEFT)  
 *# Upper Button* btn\_modify\_availability=Button(Top, width = 25, text = **"Modify Availability"**, font = (**'arial'**, 10, **'bold'**),  
 fg = **"Black"**, bg = **"#ffd587"**,  
 command = Modify\_Availability)   
 btn\_modify\_availability.pack(side = LEFT)  
 btn\_refresh=Button(Top, width = 10, text = **"Refresh (F5)"**, font = (**'arial'**, 10, **'bold'**), fg = **"White"**, bg = **"green"**,  
 command = Refresh)  
 btn\_refresh.pack(side = RIGHT)  
  
 *# ==================================LIST WIDGET========================================* scrollbary=Scrollbar(Right, orient = VERTICAL)  
 scrollbarx=Scrollbar(Right, orient = HORIZONTAL)  
 columns=(**"Number"**, **"Capacity"**, **"Type"**, **"Dept"**)  
 tree=ttk.Treeview(Right, columns = columns, selectmode = **"extended"**, height = 500,  
 yscrollcommand = scrollbary.set, xscrollcommand = scrollbarx.set)  
  
 def treeview\_sort\_column(tree, col, reverse):  
 l=[(tree.set(k, col), k) for k in tree.get\_children(**''**)]  
 l.sort(reverse = reverse)  
  
 *# rearrange items in sorted positions* for index, (val, k) in enumerate(l):  
 tree.move(k, **''**, index)  
  
 *# reverse sort next time* tree.heading(col, command = lambda \_col=col: treeview\_sort\_column(tree, \_col, not reverse))  
  
 for col in columns:  
 tree.heading(col, text = col, command = lambda \_col=col: \  
 treeview\_sort\_column(tree, \_col, False))  
 *# for TREE Scroll Bars* scrollbary.config(command = tree.yview)  
 scrollbary.pack(side = RIGHT, fill = Y)  
 scrollbarx.config(command = tree.xview)  
 scrollbarx.pack(side = BOTTOM, fill = X)  
 *# inside the tree view* tree.heading(**'Number'**, text = **"Room"**, anchor = W)  
 tree.heading(**'Capacity'**, text = **"Capacity"**, anchor = W)  
 tree.heading(**'Type'**, text = **"Type"**, anchor = W)  
 tree.heading(**'Dept'**, text = **"Dept"**, anchor = W)  
 tree.column(**'#0'**, stretch = NO, minwidth = 0, width = 0)  
 tree.column(**'#1'**, stretch = NO, minwidth = 75, width = 75)  
 tree.column(**'#2'**, stretch = NO, minwidth = 75, width = 75)  
 tree.column(**'#3'**, stretch = NO, minwidth = 75, width = 75)  
 tree.bind(**'<ButtonRelease-1>'**, click)  
 tree.pack()  
  
 *# ==================================INITIALIZATION=====================================* if \_\_name\_\_ == **'\_\_main\_\_'**:  
 View()  
 roomroot.mainloop()  
  
  
 def Class\_Creation():  
 RT=Toplevel(root)  
 RT.title(**"ECE-SPV"**)  
 RT.wm\_iconbitmap(**"ece-spv.ico"**)  
 RT.screen\_width=RT.winfo\_screenwidth()  
 RT.screen\_height=RT.winfo\_screenheight()  
 RT.width=720  
 RT.height=627  
 RT.x=(screen\_width / 2) - (width / 2)  
 RT.y=(screen\_height / 2) - (height / 2)  
 RT.geometry(**'%dx%d+%d+%d'** % (width, height, x, y))  
 RT.resizable(0, 0)  
  
 *# ==================================COMMANDS============================================* def click(event):  
 try:  
 selected=tree.focus() *# Grab record position/number* values=tree.item(selected, **'values'**) *# Grab record values* cb\_section\_code.delete(0, **'end'**)  
 cb\_subject\_code.delete(0, **'end'**)  
 cb\_section\_code.insert(0, values[0])  
 cb\_subject\_code.insert(0, values[1])  
 except IndexError: pass  
  
 def combobox\_input0():  
 ARADB()  
 cursor.execute(**"SELECT code FROM `section`"**)  
 conn.commit()  
 data=[]  
 for row in cursor.fetchall():  
 data.append(row[0])  
 return data  
 cursor.close()  
 conn.close()  
  
 def combobox\_input1():  
 ARADB()  
 cursor.execute(**"SELECT \* FROM `subject`"**)  
 conn.commit()  
 data=[]  
 for row in cursor.fetchall():  
 data.append(str(row[0]) + **' ['** + str(row[1]) + **']'**)  
 return data  
 cursor.close()  
 conn.close()  
  
 def Create():  
 if SECTION\_CODE.get() == **""** or SUBJECT\_CODE.get() == **""**:  
 txt\_result.config(text = **"Please complete the all fields!"**, fg = **"red"**)  
 else:  
 ARADB()  
 cursor.execute(**"INSERT INTO section\_subject (section\_code, subject\_code) VALUES(?, ?)"**,  
 (str(SECTION\_CODE.get()), (str(SUBJECT\_CODE.get().split(**' '**, 1)[0])),))  
 conn.commit()  
 SUBJECT\_CODE.set(**""**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Created a data!"**, fg = **"green"**)  
  
 def View():  
 tree.delete(\*tree.get\_children())  
 ARADB()  
 cursor.execute(**"SELECT \* FROM `section\_subject` ORDER BY `section\_code` ASC"**)  
 fetch=cursor.fetchall()  
 for data in fetch:  
 tree.insert(**''**, **'end'**, values = (data[0], data[1]))  
 cursor.close()  
 conn.close()  
 txt\_result.config(text = **"Successfully viewed the data from database"**, fg = **"black"**)  
  
 def Delete():  
 if SECTION\_CODE.get() == **""** or SUBJECT\_CODE.get() == **""**:  
 txt\_result.config(text = **"Please complete the all fields!"**, fg = **"red"**)  
 else:  
 ARADB()  
 cursor.execute(**"DELETE FROM 'section\_subject' WHERE section\_code = '"** + str(  
 SECTION\_CODE.get()) + **"' and subject\_code = '"** + str(SUBJECT\_CODE.get()) + **"'"**)  
 conn.commit()  
 cb\_section\_code.delete(0, **'end'**)  
 cb\_subject\_code.delete(0, **'end'**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Deleted Successfully!"**, fg = **"green"**)  
  
 def Clear():  
 result=messagebox.askquestion(**"Clear"**, **"This will clear the Class Creation table.**\n**Do you want to proceed?"**, icon = **'warning'**)  
 if result == **'yes'**:  
 ARADB()  
 cursor.execute(**"""DROP TABLE section\_subject"""**)  
 cursor.execute(**"""create table section\_subject (section\_code text NOT NULL, subject\_code text NOT NULL)"""**)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Table Cleared Successfully!"**, fg = **"green"**)  
 else:  
 pass  
  
 def Refresh():  
 RT.destroy()  
 Class\_Creation()  
  
 *# ==================================VARIABLES==========================================* SECTION\_CODE=StringVar(RT)  
 SUBJECT\_CODE=StringVar(RT)  
  
 *# ==================================FRAME==============================================* Top=Frame(RT, width = 900, height = 50, bd = 8, relief = **"raise"**, bg = **"#9EC1CF"**)  
 Top.pack(side = TOP)  
 Left=Frame(RT, width = 300, height = 500, bd = 8, relief = **"raise"**)  
 Left.pack(side = LEFT)  
 Right=Frame(RT, width = 600, height = 500, bd = 8, relief = **"raise"**)  
 Right.pack(side = RIGHT)  
 Forms=Frame(Left, width = 300, height = 450)  
 Forms.pack(side = TOP)  
 Buttons=Frame(Left, width = 300, height = 100, bd = 8, relief = **"raise"**)  
 Buttons.pack(side = BOTTOM)  
  
 *# ==================================LABEL WIDGET=======================================* txt\_title=Label(Top, width = 900, font = (**'arial'**, 24), text = **"Class Creation"**, fg = **"Black"**,  
 bg = **"#9BE0F1"**)   
 txt\_title.pack()  
 txt\_section\_code=Label(Forms, text = **"Select Section:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_section\_code.grid(row = 0, stick = **"e"**)  
 txt\_subject\_code=Label(Forms, text = **"Select Subject:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_subject\_code.grid(row = 1, stick = **"e"**)  
 txt\_result=Label(Buttons)  
 txt\_result.pack(side = TOP)  
  
 *# ==================================ENTRY WIDGET=======================================* cb\_section\_code=ttk.Combobox(Forms, textvariable = SECTION\_CODE, width = 32)  
 cb\_section\_code[**'values'**]=combobox\_input0()  
 cb\_section\_code.grid(column = 1, row = 0)  
 cb\_subject\_code=ttk.Combobox(Forms, textvariable = SUBJECT\_CODE, width = 32)  
 cb\_subject\_code[**'values'**]=combobox\_input1()  
 cb\_subject\_code.grid(column = 1, row = 1)  
  
 *# ==================================BUTTONS WIDGET=====================================* btn\_create=Button(Buttons, width = 10, text = **"Create"**, command = Create)  
 btn\_create.pack(side = LEFT)  
 btn\_view=Button(Buttons, width = 10, text = **"View"**, command = View)  
 btn\_view.pack(side = LEFT)  
 btn\_update=Button(Buttons, width = 10, text = **"Update"**, state = DISABLED)  
 btn\_update.pack(side = LEFT)  
 btn\_delete=Button(Buttons, width = 10, text = **"Delete"**, command = Delete)  
 btn\_delete.pack(side = LEFT)  
 btn\_clear=Button(Buttons, width = 10, text = **"Clear"**, fg = **"Black"**, bg = **"#F2B6AE"**, command = Clear)  
 btn\_clear.pack(side = LEFT)  
 btn\_refresh=Button(Top, width = 10, text = **"Refresh (F5)"**, font = (**'arial'**, 10, **'bold'**), fg = **"White"**, bg = **"green"**,  
 command = Refresh)  
 btn\_refresh.pack(side = RIGHT)  
  
 *# ==================================LIST WIDGET========================================* scrollbary=Scrollbar(Right, orient = VERTICAL)  
 scrollbarx=Scrollbar(Right, orient = HORIZONTAL)  
 columns=(**"Section"**, **"Subject"**)  
 tree=ttk.Treeview(Right, columns = columns, selectmode = **"extended"**, height = 500,  
 yscrollcommand = scrollbary.set, xscrollcommand = scrollbarx.set)  
  
 def treeview\_sort\_column(tree, col, reverse):  
 l=[(tree.set(k, col), k) for k in tree.get\_children(**''**)]  
 l.sort(reverse = reverse)  
  
 *# rearrange items in sorted positions* for index, (val, k) in enumerate(l):  
 tree.move(k, **''**, index)  
  
 *# reverse sort next time* tree.heading(col, command = lambda \_col=col: treeview\_sort\_column(tree, \_col, not reverse))  
  
 for col in columns:  
 tree.heading(col, text = col, command = lambda \_col=col: \  
 treeview\_sort\_column(tree, \_col, False))  
 *# for TREE Scroll Bars* scrollbary.config(command = tree.yview)  
 scrollbary.pack(side = RIGHT, fill = Y)  
 scrollbarx.config(command = tree.xview)  
 scrollbarx.pack(side = BOTTOM, fill = X)  
 *# inside the tree view* tree.heading(**'Section'**, text = **"Section"**, anchor = W)  
 tree.heading(**'Subject'**, text = **"Subject"**, anchor = W)  
 tree.column(**'#0'**, stretch = NO, minwidth = 0, width = 0)  
 tree.column(**'#1'**, stretch = NO, minwidth = 150, width = 150)  
 tree.bind(**'<ButtonRelease-1>'**, click)  
 tree.pack()  
  
 if \_\_name\_\_ == **'\_\_main\_\_'**:  
 View()  
 RT.mainloop()  
  
  
 def Instructor():  
 instroot=Toplevel(root)  
 instroot.title(**"ECE-SPV"**)  
 instroot.wm\_iconbitmap(**"ece-spv.ico"**)  
 screen\_width=instroot.winfo\_screenwidth()  
 screen\_height=instroot.winfo\_screenheight()  
 width=900  
 height=627  
 x=(screen\_width / 2) - (width / 2)  
 y=(screen\_height / 2) - (height / 2)  
 instroot.geometry(**'%dx%d+%d+%d'** % (width, height, x, y))  
 instroot.resizable(0, 0)  
  
 *# ==================================COMMANDS============================================* def click(event):  
 try:  
 selected=tree.focus() *# Grab record position/number* values=tree.item(selected, **'values'**) *# Grab record values* cb\_number.delete(0, **'end'**)  
 e\_name.delete(0, **'end'**)  
 e\_max\_hours.delete(0, **'end'**)  
 e\_seniority.delete(0, **'end'**)  
 cb\_number.insert(0, values[0])  
 e\_name.insert(0, values[1])  
 e\_max\_hours.insert(0, values[2])  
 e\_seniority.insert(0, values[3])  
 except IndexError: pass  
  
 def combobox\_input():  
 ARADB()  
 cursor.execute(**"SELECT number FROM `instructor`"**)  
 conn.commit()  
 data=[]  
 for row in cursor.fetchall():  
 data.append(row[0])  
 return data  
 cursor.close()  
 conn.close()  
  
 def Create():  
 if NUMBER.get() == **""** or NAME.get() == **""** or MAX\_HOURS.get() == 0 or SENIORITY.get() == 0 or MAX\_HOURS.get() == **""** or SENIORITY.get() == **""**:  
 txt\_result.config(text = **"Please complete all fields!"**, fg = **"red"**)  
 else:  
 if len(NAME.get()) <= 30:  
 ARADB()  
 cursor.execute(**"INSERT INTO `instructor` (number, name, max\_hours, seniority) VALUES(?, ?, ?, ?)"**,  
 (str(NUMBER.get()), (str(NAME.get())), (str(MAX\_HOURS.get())), (str(SENIORITY.get())),))  
 conn.commit()  
 NUMBER.set(**""**)  
 NAME.set(**""**)  
 MAX\_HOURS.set(**""**)  
 SENIORITY.set(**""**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Created a data!"**, fg = **"green"**)  
 else:  
 result=messagebox.showinfo(**"String Length Limit: 30"**, **"The name length exceeds the limit."**, icon = **'info'**)  
 if result == **'ok'**:  
 pass  
  
 def View():  
 tree.delete(\*tree.get\_children())  
 ARADB()  
 cursor.execute(**"SELECT \* FROM `instructor` ORDER BY `name` ASC"**)  
 fetch=cursor.fetchall()  
 for data in fetch:  
 tree.insert(**''**, **'end'**, values = (data[0], data[1], data[2], data[3]))  
 cursor.close()  
 conn.close()  
 txt\_result.config(text = **"Successfully viewed the data from database"**, fg = **"black"**)  
  
 def Update():  
 if NUMBER.get() == **""** or NAME.get() == **""** or MAX\_HOURS == 0 or SENIORITY == 0 or MAX\_HOURS == **""** or SENIORITY == **""**:  
 txt\_result.config(text = **"Please complete all fields!"**, fg = **"red"**)  
 else:  
 ARADB()  
 cursor.execute(  
 **"UPDATE `instructor` SET name = :e\_name, max\_hours = :e\_max\_hours, seniority = :e\_seniority WHERE number = :cb\_number"""**, {  
 **'e\_name'**: e\_name.get(), **'e\_max\_hours'**: e\_max\_hours.get(), **'e\_seniority'**: e\_seniority.get(), **'cb\_number'**: cb\_number.get()})  
 conn.commit()  
 cb\_number.delete(0, **'end'**)  
 e\_name.delete(0, **'end'**)  
 e\_max\_hours.delete(0, **'end'**)  
 e\_seniority.delete(0, **'end'**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Updated Successfully!"**, fg = **"green"**)  
  
 def Delete():  
 if NUMBER.get() == **""**:  
 txt\_result.config(text = **"Please enter an ID Number!"**, fg = **"red"**)  
 else:  
 ARADB()  
 cursor.execute(**"DELETE FROM 'instructor' WHERE number = '"** + str(NUMBER.get()) + **"'"**)  
 conn.commit()  
 cb\_number.delete(0, **'end'**)  
 e\_name.delete(0, **'end'**)  
 e\_max\_hours.delete(0, **'end'**)  
 e\_seniority.delete(0, **'end'**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Deleted Successfully!"**, fg = **"green"**)  
  
 def Clear():  
 result=messagebox.askquestion(**"Clear"**, **"This will clear the Instructor table.**\n**Do you want to proceed?"**, icon = **'warning'**)  
 if result == **'yes'**:  
 ARADB()  
 cursor.execute(**"""DROP TABLE instructor"""**)  
 cursor.execute(**"""create table instructor (number text PRIMARY KEY, name text, max\_units integer NOT NULL, seniority integer NOT NULL)"""**)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Table Cleared Successfully!"**, fg = **"green"**)  
 else:  
 pass  
  
 def Modify\_Availability():  
 IA=Toplevel(instroot)  
 IA.title(**"ECE-SPV"**)  
 IA.wm\_iconbitmap(**"ece-spv.ico"**)  
 IA.screen\_width=IA.winfo\_screenwidth()  
 IA.screen\_height=IA.winfo\_screenheight()  
 IA.width=720  
 IA.height=627  
 IA.x=(screen\_width / 2) - (width / 2)  
 IA.y=(screen\_height / 2) - (height / 2)  
 IA.geometry(**'%dx%d+%d+%d'** % (width, height, x, y))  
 IA.resizable(0, 0)  
  
 *# ==================================COMMANDS============================================* class DBMgr:  
 def \_\_init\_\_(self):  
 self.\_conn=sqlite.connect(**'ARADB.db'**) *# sql connector* self.\_cursor=self.\_conn.cursor() *# sql cursor* self.\_times=self.\_select\_times() *# select time sql cursor* self.\_timepoints=self.\_select\_timepoints() *# select timepoints sql cursor* self.\_meetingTimes=self.\_select\_meeting\_times() *# select meeting time sql cursor* self.\_instructors=self.\_select\_instructors() *# select instructor ql cursor* self.\_subjects=self.\_select\_subjects() *# select subject sql cursor* self.\_rooms=self.\_select\_rooms() *# select room sql cursor* self.\_sections=self.\_select\_sections() *# select section sql cursor* self.\_numberOfClasses=0 *# initial number of classes* for i in range(0, len(self.\_sections)):  
 self.\_numberOfClasses+=len(  
 self.\_sections[i].get\_subjects()) *# auto-itereation of number of classes creation  
  
 # select section sql command* def \_select\_sections(self):  
 self.\_cursor.execute(**"SELECT \* FROM section"**)  
 sections=self.\_cursor.fetchall()  
 returnSections=[]  
 for i in range(0, len(sections)):  
 returnSections.append(Section(sections[i][0], self.\_select\_section\_subjects(sections[i][0]),  
 self.\_select\_section\_availability(sections[i][0])))  
 return returnSections  
   
 *# select subject sql command* def \_select\_subjects(self):  
 self.\_cursor.execute(**"SELECT \* FROM subject"**)  
 subjects=self.\_cursor.fetchall()  
 returnSubjects=[]  
 for i in range(0, len(subjects)):  
 returnSubjects.append(  
 Subject(subjects[i][0], subjects[i][1], subjects[i][2], subjects[i][3], subjects[i][4],  
 subjects[i][5], subjects[i][6], subjects[i][7], self.\_select\_subject\_instructors(  
 subjects[i][0])))   
 return returnSubjects  
  
 *# select instructor sql command* def \_select\_instructors(self):  
 self.\_cursor.execute(**"SELECT \* FROM instructor"**)  
 instructors=self.\_cursor.fetchall()  
 returnInstructors=[]  
 for i in range(0, len(instructors)):  
 returnInstructors.append(  
 Instructor(instructors[i][0], instructors[i][1], instructors[i][2], instructors[i][3],  
 self.\_select\_instructor\_availability(instructors[i][0]),  
 self.\_select\_instructor\_availability\_start(instructors[i][0]),  
 self.\_select\_instructor\_availability\_end(instructors[i][0]),  
 self.\_select\_instructor\_overtime(instructors[i][0])))  
 return returnInstructors  
  
 *# select room sql command* def \_select\_rooms(self):  
 self.\_cursor.execute(**"SELECT \* FROM room"**)  
 rooms=self.\_cursor.fetchall()  
 returnRooms=[]  
 for i in range(0, len(rooms)):  
 returnRooms.append(Room(rooms[i][0], rooms[i][1], rooms[i][2], rooms[i][3],  
 self.\_select\_room\_availability(  
 rooms[i][  
 0])))   
 return returnRooms  
  
 *# select meeting time sql command* def \_select\_meeting\_times(self):  
 self.\_cursor.execute(**"SELECT \* FROM meeting\_time"**)  
 meetingTimes=self.\_cursor.fetchall()  
 returnMeetingTimes=[]  
 for i in range(0, len(meetingTimes)):  
 returnMeetingTimes.append(  
 MeetingTime(meetingTimes[i][0], meetingTimes[i][1], meetingTimes[i][2],  
 meetingTimes[i][3], meetingTimes[i][4],  
 self.\_select\_rest\_time(meetingTimes[i][0]),  
 self.\_select\_break\_time(meetingTimes[i][0])))   
 return returnMeetingTimes  
  
 *# select time sql command* def \_select\_times(self):  
 self.\_cursor.execute(**"SELECT \* FROM time"**)  
 times=self.\_cursor.fetchall()  
 returnTimes=[]  
 for i in range(0, len(times)):  
 returnTimes.append(Time(times[i][0]))  
 return returnTimes  
  
 *# select timepoint sql command* def \_select\_timepoints(self):  
 self.\_cursor.execute(**"SELECT \* FROM time\_point"**)  
 timepoints=self.\_cursor.fetchall()  
 returnTimePoints=[]  
 for i in range(0, len(timepoints)):  
 returnTimePoints.append(TimePoint(timepoints[i][0],  
 self.\_select\_point\_block(timepoints[i][0]),  
 self.\_select\_phantom\_block(timepoints[i][0])))  
 return returnTimePoints  
  
 *# GETTING VALUE* def \_select\_section\_subjects(self, sectionCode):  
 self.\_cursor.execute(**"SELECT \* FROM section\_subject where section\_code == '"** + sectionCode + **"'"**)  
 dbSubjectCodes=self.\_cursor.fetchall()  
 subjectCodes=[]  
 for i in range(0, len(dbSubjectCodes)):  
 subjectCodes.append(dbSubjectCodes[i][1])  
 returnValue=[]  
 for i in range(0, len(self.\_subjects)):  
 if self.\_subjects[i].get\_code() in subjectCodes:  
 returnValue.append(self.\_subjects[i])  
 return returnValue  
  
 def \_select\_subject\_instructors(self, subjectCode):  
 self.\_cursor.execute(**"SELECT \* FROM subject\_instructor where subject\_code == '"** + subjectCode + **"'"**)  
 dbInstructorNumbers=self.\_cursor.fetchall()  
 instructorNumbers=[]  
 for i in range(0, len(dbInstructorNumbers)): instructorNumbers.append(dbInstructorNumbers[i][1])  
 returnValue=[]  
 for i in range(0, len(self.\_instructors)):  
 if self.\_instructors[i].get\_number() in instructorNumbers:  
 returnValue.append(self.\_instructors[i])  
 return returnValue  
  
 def \_select\_instructor\_availability(self, instructor):  
 self.\_cursor.execute(  
 **"SELECT \* from instructor\_availability where instructor\_id == '"** + instructor + **"'"**)  
 instructorMTsRS=self.\_cursor.fetchall()  
 instructorMTs=[]  
 for i in range(0, len(instructorMTsRS)):  
 instructorMTs.append(instructorMTsRS[i][1])  
 instructorAvailability=list()  
 for i in range(0, len(self.\_meetingTimes)):  
 if self.\_meetingTimes[i].get\_id() in instructorMTs:  
 instructorAvailability.append(self.\_meetingTimes[i].get\_id())  
 return instructorAvailability  
  
 def \_select\_instructor\_availability\_start(self, instructor):  
 self.\_cursor.execute(  
 **"SELECT \* from instructor\_availability where instructor\_id == '"** + instructor + **"'"**)  
 instructorMTsRS=self.\_cursor.fetchall()  
 instructorMTsSTART=[]  
 for i in range(0, len(instructorMTsRS)):  
 instructorMTsSTART.append(instructorMTsRS[i][2])  
 instructorAvailabilitySTART=list()  
 for i in range(0, len(self.\_timepoints)):  
 if self.\_timepoints[i].get\_point() in instructorMTsSTART:  
 instructorAvailabilitySTART.append(self.\_timepoints[i].get\_point())  
 return instructorAvailabilitySTART  
  
 def \_select\_instructor\_availability\_end(self, instructor):  
 self.\_cursor.execute(  
 **"SELECT \* from instructor\_availability where instructor\_id == '"** + instructor + **"'"**)  
 instructorMTsRS=self.\_cursor.fetchall()  
 instructorMTsEND=[]  
 for i in range(0, len(instructorMTsRS)):  
 instructorMTsEND.append(instructorMTsRS[i][3])  
 instructorAvailabilityEND=list()  
 for i in range(0, len(self.\_timepoints)):  
 if self.\_timepoints[i].get\_point() in instructorMTsEND:  
 instructorAvailabilityEND.append(self.\_timepoints[i].get\_point())  
 return instructorAvailabilityEND  
  
 def \_select\_instructor\_overtime(self, instructor):  
 self.\_cursor.execute(  
 **"SELECT \* from instructor\_overtime where instructor\_id == '"** + instructor + **"'"**)  
 instructorMTsRS=self.\_cursor.fetchall()  
 instructorMTs=[]  
 for i in range(0, len(instructorMTsRS)):  
 instructorMTs.append(instructorMTsRS[i][1])  
 instructorOvertime=list()  
 for i in range(0, len(self.\_meetingTimes)):  
 if self.\_meetingTimes[i].get\_id() in instructorMTs:  
 instructorOvertime.append(self.\_meetingTimes[i].get\_id())  
 return instructorOvertime  
  
 def \_select\_room\_availability(self, room):  
 self.\_cursor.execute(**"SELECT \* from room\_availability where room\_number = '"** + room + **"'"**)  
 roomMTsRS=self.\_cursor.fetchall()  
 roomMTs=[]  
 for i in range(0, len(roomMTsRS)): roomMTs.append(roomMTsRS[i][1])  
 roomAvailability=list()  
 for i in range(0, len(self.\_meetingTimes)):  
 if self.\_meetingTimes[i].get\_id() in roomMTs:  
 roomAvailability.append(self.\_meetingTimes[i].get\_id())  
 return roomAvailability  
  
 def \_select\_section\_availability(self, section):  
 self.\_cursor.execute(**"SELECT \* from section\_availability where section\_code = '"** + section + **"'"**)  
 sectionMTsRS=self.\_cursor.fetchall()  
 sectionMTs=[]  
 for i in range(0, len(sectionMTsRS)): sectionMTs.append(sectionMTsRS[i][1])  
 sectionAvailability=list()  
 for i in range(0, len(self.\_meetingTimes)):  
 if self.\_meetingTimes[i].get\_id() in sectionMTs:  
 sectionAvailability.append(self.\_meetingTimes[i].get\_id())  
 return sectionAvailability  
  
 def \_select\_rest\_time(self, meeting\_time):  
 self.\_cursor.execute(**"SELECT \* from rest\_time where meeting\_time\_id = '"** + meeting\_time + **"'"**)  
 timeMTsRS=self.\_cursor.fetchall()  
 timeMTs=[]  
 for i in range(0, len(timeMTsRS)): timeMTs.append(timeMTsRS[i][1])  
 restTime=list()  
 for i in range(0, len(self.\_times)):  
 if self.\_times[i].get\_block() in timeMTs:  
 restTime.append(self.\_times[i].get\_block())  
 return restTime  
  
 def \_select\_break\_time(self, meeting\_time):  
 self.\_cursor.execute(**"SELECT \* from break\_time where meeting\_time\_id = '"** + meeting\_time + **"'"**)  
 timeMTsRS=self.\_cursor.fetchall()  
 timeMTs=[]  
 for i in range(0, len(timeMTsRS)): timeMTs.append(timeMTsRS[i][1])  
 breakTime=list()  
 for i in range(0, len(self.\_times)):  
 if self.\_times[i].get\_block() in timeMTs:  
 breakTime.append(self.\_times[i].get\_block())  
 return breakTime  
  
 def \_select\_point\_block(self, point):  
 self.\_cursor.execute(**"SELECT \* from point\_block where time\_point == '"** + point + **"'"**)  
 pointBlocksRS=self.\_cursor.fetchall()  
 pointBlocks=[]  
 for i in range(0, len(pointBlocksRS)): pointBlocks.append(pointBlocksRS[i][1])  
 returnValue=list()  
 for i in range(0, len(self.\_times)):  
 if self.\_times[i].get\_block() in pointBlocks:  
 returnValue.append(self.\_times[i].get\_block())  
 return returnValue  
  
 def \_select\_phantom\_block(self, point):  
 self.\_cursor.execute(**"SELECT \* from phantom\_block where time\_point == '"** + point + **"'"**)  
 pointBlocksRS=self.\_cursor.fetchall()  
 pointBlocks=[]  
 for i in range(0, len(pointBlocksRS)): pointBlocks.append(pointBlocksRS[i][1])  
 returnValue=list()  
 for i in range(0, len(self.\_times)):  
 if self.\_times[i].get\_block() in pointBlocks:  
 returnValue.append(self.\_times[i].get\_block())  
 return returnValue  
  
 def get\_sections(self):  
 return self.\_sections *# get item to section table* def get\_subjects(self):  
 return self.\_subjects *# get item to subject table* def get\_instructors(self):  
 return self.\_instructors *# get item to instructor table* def get\_meetingTimes(self):  
 return self.\_meetingTimes *# get item to meeting time table* def get\_times(self):  
 return self.\_times *# get item to time table* def get\_timepoints(self):  
 return self.\_timepoints *# get item to time table* def get\_rooms(self):  
 return self.\_rooms *# get item to room table* def get\_numberOfClasses(self):  
 return self.\_numberOfClasses *# get item to number of class table* class Schedule:  
 def \_\_init\_\_(self):  
 self.\_data=dbMgr  
 self.\_classes=[]  
 self.\_conflicts=[]  
 self.\_fitness=-1  
 self.\_classNumb=0  
 self.\_isFitnessChanged=True  
 self.\_generationNumber=0  
  
 def get\_classes(self):  
 self.\_isFitnessChanged=True  
 return self.\_classes  
  
 def get\_conflicts(self):  
 return self.\_conflicts  
  
 def get\_fitness(self):  
 if (self.\_isFitnessChanged == True):  
 self.\_fitness=self.calculate\_fitness()  
 self.\_isFitnessChanged=False  
 return self.\_fitness  
  
 def initialize(self):  
 sections=dbMgr.get\_sections() *# point to section table* meetingTimes=dbMgr.get\_meetingTimes()  
 rooms=dbMgr.get\_rooms()  
 classes=self.\_classes  
 PR=list()  
 instructorList=list()  
 MT1=list()  
 MT1p5=list()  
 MT2=list()  
 MT3=list()  
 MT4=list()  
 MT1v2=list()  
 MT1p5v2=list()  
 MT2v2=list()  
 MT3v2=list()  
 for RM in range(0, len(rooms)):  
 if (rooms[RM].get\_type() == **'PR'**):  
 PR.append(rooms[RM])  
 for MT in range(0, len(meetingTimes)):  
 if (meetingTimes[MT].get\_MThour() == 1):  
 MT1.append(meetingTimes[MT])  
 MT1v2.append(meetingTimes[MT])  
 if (meetingTimes[MT].get\_MThour() == 1.5):  
 MT1p5.append(meetingTimes[MT])  
 MT1p5v2.append(meetingTimes[MT])  
 if (meetingTimes[MT].get\_MThour() == 2):  
 MT2.append(meetingTimes[MT])  
 MT2v2.append(meetingTimes[MT])  
 if (meetingTimes[MT].get\_MThour() == 3):  
 MT3.append(meetingTimes[MT])  
 MT3v2.append(meetingTimes[MT])  
 if (meetingTimes[MT].get\_MThour() == 4):  
 MT4.append(meetingTimes[MT])  
 for i in range(0, len(sections)): *# List all the items per coloumn* subjects=sections[i].get\_subjects() *# point to subject table* for j in range(0, len(subjects)): *# List all the items per coloumn* newClass=Class(self.\_classNumb, sections[i], subjects[j]) *# Create a new class* self.\_classNumb+=1 *# Iterate the class creation* newClass.set\_instructor(subjects[j].get\_instructors()[rnd.randrange(0, len(subjects[  
 j].get\_instructors()))]) *# Pick a random instructor taht can teach the assigned subject for the new class* if (subjects[j].get\_compatibility() == **'PR'**):  
 newClass.set\_room(  
 PR[rnd.randrange(0, len(PR))]) *# Pick a random room in database for the new class* else:  
 newClass.set\_room(  
 rooms[rnd.randrange(0,  
 len(rooms))]) *# Pick a random room in database for the new class"""* def wholetwohours():  
 newClass.set\_meetingTime(MT2[rnd.randrange(0, len(MT2))])  
 newClass.set\_meetingTime1(dbMgr.get\_meetingTimes()[0])  
  
 def wholethreehours():  
 newClass.set\_meetingTime(MT3[rnd.randrange(0, len(MT3))])  
 newClass.set\_meetingTime1(dbMgr.get\_meetingTimes()[0])  
  
 def wholefourhours():  
 newClass.set\_meetingTime(MT4[rnd.randrange(0, len(MT4))])  
 newClass.set\_meetingTime1(dbMgr.get\_meetingTimes()[0])  
  
 def splittwohours():  
 newClass.set\_meetingTime(MT1[rnd.randrange(0, len(MT1))])  
 newClass.set\_meetingTime1(MT1v2[rnd.randrange(0, len(MT1v2))])  
  
 def splitthreehours():  
 newClass.set\_meetingTime(MT1p5[rnd.randrange(0, len(MT1p5))])  
 newClass.set\_meetingTime1(MT1p5v2[rnd.randrange(0, len(MT1p5v2))])  
  
 def splitfourhours():  
 newClass.set\_meetingTime(MT2[rnd.randrange(0, len(MT2))])  
 newClass.set\_meetingTime1(MT2v2[rnd.randrange(0, len(MT2v2))])  
  
 def call\_funcs\_randomly(funcs):  
 shuffle(funcs)  
 for func in funcs:  
 func()  
  
 *# with mixed Split Time* if (subjects[j].get\_numbHour() == 1):  
 newClass.set\_meetingTime(MT1[rnd.randrange(0, len(MT1))])  
 newClass.set\_meetingTime1(dbMgr.get\_meetingTimes()[0])  
 elif (subjects[j].get\_numbHour() == 2):  
 call\_funcs\_randomly([wholetwohours, splittwohours])  
 elif (subjects[j].get\_numbHour() == 3):  
 call\_funcs\_randomly([wholethreehours, splitthreehours])  
 elif (subjects[j].get\_numbHour() == 4):  
 call\_funcs\_randomly([wholefourhours, splitfourhours])  
 elif (subjects[j].get\_numbHour() == 6):  
 newClass.set\_meetingTime(MT3[rnd.randrange(0, len(MT3))])  
 newClass.set\_meetingTime1(MT3v2[rnd.randrange(0, len(MT3v2))])  
 else:  
 newClass.set\_meetingTime(  
 dbMgr.get\_meetingTimes()[rnd.randrange(0, len(dbMgr.get\_meetingTimes()))])  
 newClass.set\_meetingTime1(  
 dbMgr.get\_meetingTimes()[rnd.randrange(0, len(dbMgr.get\_meetingTimes()))])  
 self.\_classes.append(newClass) *# Add result as new class* return self  
  
 def calculate\_fitness(self):  
 self.\_conflicts=[]  
 classes=self.get\_classes()  
 instructorNames=list()  
 instructor\_Hours=list()  
 for i in range(0, len(classes)):  
 instructorNames.append(classes[i].get\_instructor().get\_name())  
 subjectHours=list()  
 if (classes[i].get\_meetingTime1().get\_id() != **'NULL'**):  
 if (classes[i].get\_meetingTime().get\_id() == classes[i].get\_meetingTime1().get\_id()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(Conflict(Conflict.ConflictType.SAME\_MTS, conflictBetweenClasses))  
 if DISABLE\_UNEQUAL\_SPLIT == True:  
 if (classes[i].get\_meetingTime().get\_MThour() != classes[  
 i].get\_meetingTime1().get\_MThour()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.UNEQUAL\_SPLIT, conflictBetweenClasses))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 i].get\_meetingTime1().get\_restTime()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.OVERLAP\_MTS, conflictBetweenClasses))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 i].get\_meetingTime().get\_restTime()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.OVERLAP\_MTS, conflictBetweenClasses))  
 if DISABLE\_CASUAL\_SPLITTING == True:  
 *# Disable Same Day Split Time* if (classes[i].get\_meetingTime().get\_day() == classes[i].get\_meetingTime1().get\_day()):  
 if (classes[i].get\_meetingTime().get\_MThour() != classes[  
 i].get\_subject().get\_numbHour()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.CASUAL\_SPLITTING, conflictBetweenClasses))  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[i].get\_meetingTime1().get\_MThour()) !=  
 classes[i].get\_subject().get\_numbHour()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.WRONG\_MTHOUR, conflictBetweenClasses))  
 if ENABLE\_NUMB\_OF\_STUDENTS == True:  
 if (classes[i].get\_room().get\_seatingCapacity() < classes[  
 i].get\_subject().get\_maxNumbOfStudents()):  
 seatingCapacityConflict=list()  
 seatingCapacityConflict.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.NUMB\_OF\_STUDENTS, seatingCapacityConflict))  
 *# Disable to assign Labs on VR* if LAB\_ON\_VR == False:  
 if (classes[i].get\_subject().get\_compatibility() == **'PR'**):  
 if (classes[i].get\_room().get\_type() != **'PR'**):  
 roomCompatibilityConflict=list()  
 roomCompatibilityConflict.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.LAB\_ON\_VR, roomCompatibilityConflict))  
 *# Disable cross department room utility* if DISABLE\_XDEPT\_ROOM\_UTILITY == True:  
 if (classes[i].get\_room().get\_type() != **'VR'**):  
 if (classes[i].get\_subject().get\_dept() != classes[i].get\_room().get\_dept()):  
 roomUtilityConflict=list()  
 roomUtilityConflict.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.XDEPT\_ROOM\_UTILITY, roomUtilityConflict))  
  
 *# if ENABLE\_INSTRUCTOR\_OVERTIME == True:* if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 0* if (classes[i].get\_meetingTime().get\_id() in classes[  
 i].get\_instructor().get\_overtime()): *# This identify the schedules inside the overtime* if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_OVERTIME, conflictBetweenClasses))  
 else:  
 *# 0* if (classes[i].get\_meetingTime().get\_id() in classes[i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_instructor().get\_availability()): *# This removes the schedules out side the official time and overtime* conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_OVERTIME, conflictBetweenClasses))  
 *# 1* if (classes[i].get\_meetingTime1().get\_id() in classes[i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime1().get\_id() not in classes[  
 i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_OVERTIME, conflictBetweenClasses))  
 *# Limit Instructor Availibilty* if LIMIT\_INSTRUCTOR\_AVAILABILITY == True:  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_AVAILABILITY,  
 conflictBetweenClasses))  
 else:  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_AVAILABILITY,  
 conflictBetweenClasses))  
 *# 1* if (classes[i].get\_meetingTime1().get\_id() not in classes[  
 i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime1().get\_id() not in classes[  
 i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_AVAILABILITY,  
 conflictBetweenClasses))  
 *# Limit Room Availibilty* if LIMIT\_ROOM\_AVAILABILITY == True:  
 if (classes[i].get\_room().get\_type() != **'VR'**):  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_room().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_AVAILABILITY, conflictBetweenClasses))  
 else:  
 *# 1* if (classes[i].get\_meetingTime1().get\_id() not in classes[  
 i].get\_room().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_AVAILABILITY, conflictBetweenClasses))  
 *# Limit Section Availibilty* if LIMIT\_SECTION\_AVAILABILITY == True:  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_section().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_AVAILABILITY, conflictBetweenClasses))  
 else:  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_section().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_AVAILABILITY, conflictBetweenClasses))  
 *# 1* if (classes[i].get\_meetingTime1().get\_id() not in classes[  
 i].get\_section().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_AVAILABILITY, conflictBetweenClasses))  
  
 for j in range(0, len(classes)):  
 *# Class Optimization* if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day() and  
 classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[  
 j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 else:  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day() and  
 classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[  
 j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 *# 11* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime1().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime1().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[  
 j].get\_meetingTime1().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime1().get\_breakTime()):  
 if ((classes[i].get\_meetingTime1().get\_MThour() + classes[  
 j].get\_meetingTime1().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 *# 10* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime1().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime1().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime1().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime1().get\_MThour() + classes[  
 j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 *# 01* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime1().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[  
 j].get\_meetingTime1().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 *# Instructor Optimization* if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[  
 j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 else:  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[  
 j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# 11* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime1().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime1().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[  
 j].get\_meetingTime1().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime1().get\_breakTime()):  
 if ((classes[i].get\_meetingTime1().get\_MThour() + classes[  
 j].get\_meetingTime1().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# 01* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime1().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime1().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime1().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime1().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[  
 j].get\_meetingTime1().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# 10* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime1().get\_MThour() + classes[  
 j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# Room Optimization* if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day() and  
 classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 else:  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day() and  
 classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 *# 11* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime1().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime1().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[  
 j].get\_meetingTime1().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 *# 01* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime1().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime1().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime1().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 *# 10* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 if (j >= i):  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_meetingTime().get\_id() == classes[j].get\_meetingTime().get\_id()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, roomBookingConflict))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 else:  
 *# 00* if (classes[i].get\_meetingTime().get\_id() == classes[j].get\_meetingTime().get\_id()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, roomBookingConflict))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# 11* if (classes[i].get\_meetingTime1().get\_id() == classes[j].get\_meetingTime1().get\_id()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, roomBookingConflict))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# 01* if (classes[i].get\_meetingTime().get\_id() == classes[j].get\_meetingTime1().get\_id()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, roomBookingConflict))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# 10* if (classes[i].get\_meetingTime1().get\_id() == classes[j].get\_meetingTime().get\_id()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, roomBookingConflict))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# Disable Dual Class Type* if DISABLE\_MIXED\_TYPE == True:  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime().get\_day() == classes[  
 j].get\_meetingTime().get\_day()):   
 if (classes[i].get\_room().get\_type() != classes[  
 j].get\_room().get\_type()):   
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 roomBookingConflict)) *# Room Type Conflict* else:  
 *# 00* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime().get\_day() == classes[  
 j].get\_meetingTime().get\_day()):   
 if (classes[i].get\_room().get\_type() != classes[  
 j].get\_room().get\_type()):   
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 roomBookingConflict)) *# Room Type Conflict  
 # 11* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime1().get\_day() == classes[  
 j].get\_meetingTime1().get\_day()):   
 if (classes[i].get\_room().get\_type() != classes[  
 j].get\_room().get\_type()):   
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 roomBookingConflict)) *# Room Type Conflict  
 # 01* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime().get\_day() == classes[  
 j].get\_meetingTime1().get\_day()):   
 if (classes[i].get\_room().get\_type() != classes[  
 j].get\_room().get\_type()):   
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 roomBookingConflict)) *# Room Type Conflict  
 # 10* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime1().get\_day() == classes[  
 j].get\_meetingTime().get\_day()):   
 if (classes[i].get\_room().get\_type() != classes[  
 j].get\_room().get\_type()):   
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 roomBookingConflict)) *# Room Type Conflict* if (classes[j].get\_instructor().get\_name() == instructorNames[i]):  
 subjectHours.append(float(classes[j].get\_subject().get\_numbHour()))  
 *# Enable Unit Limit* instructor\_Hours.append(subjectHours)  
 if DISABLE\_INSTRUCTOR\_OVERLOAD == True:  
 if (classes[i].get\_instructor().get\_max\_hours() < sum(instructor\_Hours[i])):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_OVERLOAD, instructorBookingConflict))  
 return 1 / ((1.0 \* len(self.\_conflicts) + 1))  
  
 def \_\_str\_\_(self):  
 returnValue=**""** for i in range(0, len(self.\_classes) - 1):  
 returnValue+=str(self.\_classes[i]) + **", "** returnValue+=str(self.\_classes[len(self.\_classes) - 1])  
 return returnValue  
  
 class Population:  
 def \_\_init\_\_(self, size):  
 self.\_size=size  
 self.\_data=dbMgr  
 self.\_schedules=[]  
 for i in range(0, size): self.\_schedules.append(Schedule().initialize())  
  
 def get\_size(self): return self.\_size  
  
 def get\_schedules(self): return self.\_schedules  
  
 class GeneticAlgorithm:  
 def evolve(self, population):  
 return self.\_mutate\_population(self.\_crossover\_population(population))  
  
 def \_crossover\_population(self, pop):  
 crossover\_pop=Population(0)  
 for i in range(NUMB\_OF\_ELITE\_SCHEDULES):  
 crossover\_pop.get\_schedules().append(pop.get\_schedules()[i])  
 i=NUMB\_OF\_ELITE\_SCHEDULES  
 while i < POPULATION\_SIZE:  
 schedule1=self.\_select\_tournament\_population(pop).get\_schedules()[0]  
 schedule2=self.\_select\_tournament\_population(pop).get\_schedules()[0]  
 crossover\_pop.get\_schedules().append(self.\_crossover\_schedule(schedule1, schedule2))  
 i+=1  
 return crossover\_pop  
  
 def \_mutate\_population(self, population):  
 for i in range(NUMB\_OF\_ELITE\_SCHEDULES, POPULATION\_SIZE):  
 self.\_mutate\_schedule(population.get\_schedules()[i])  
 return population  
  
 def \_crossover\_schedule(self, schedule1, schedule2):  
 crossoverSchedule=Schedule().initialize()  
 for i in range(0, len(crossoverSchedule.get\_classes())):  
 if (rnd.random() > 0.5):  
 crossoverSchedule.get\_classes()[i]=schedule1.get\_classes()[i]  
 else:  
 crossoverSchedule.get\_classes()[i]=schedule2.get\_classes()[i]  
 return crossoverSchedule  
  
 def \_mutate\_schedule(self, mutateSchedule):  
 schedule=Schedule().initialize()  
 for i in range(0, len(mutateSchedule.get\_classes())):  
 if (MUTATION\_RATE > rnd.random()): mutateSchedule.get\_classes()[i]=schedule.get\_classes()[i]  
 return mutateSchedule  
  
 def \_select\_tournament\_population(self, pop):  
 tournament\_pop=Population(0)  
 i=0  
 while i < TOURNAMENT\_SELECTION\_SIZE:  
 tournament\_pop.get\_schedules().append(  
 pop.get\_schedules()[rnd.randrange(0, int(POPULATION\_SIZE))])  
 i+=1  
 tournament\_pop.get\_schedules().sort(key = lambda x: x.get\_fitness(), reverse = True)  
 return tournament\_pop  
  
 class Section:  
 def \_\_init\_\_(self, code, subjects, availability):  
 self.\_code=code  
 self.\_subjects=subjects  
 self.\_availability=availability  
  
 def get\_code(self): return self.\_code  
  
 def get\_subjects(self): return self.\_subjects  
  
 def get\_availability(self): return self.\_availability  
  
 class Subject:  
 def \_\_init\_\_(self, code, description, units, numbHour, compatibility, curriculum, maxNumbOfStudents,  
 dept,  
 instructors):  
 self.\_code=code  
 self.\_description=description  
 self.\_units=units  
 self.\_numbHour=numbHour  
 self.\_compatibility=compatibility  
 self.\_curriculum=curriculum  
 self.\_maxNumbOfStudents=maxNumbOfStudents  
 self.\_dept=dept  
 self.\_instructors=instructors  
  
 def get\_code(self): return self.\_code  
  
 def get\_description(self): return self.\_description  
  
 def get\_units(self): return self.\_units  
  
 def get\_numbHour(self): return self.\_numbHour  
  
 def get\_compatibility(self): return self.\_compatibility  
  
 def get\_curriculum(self): return self.\_curriculum  
  
 def get\_maxNumbOfStudents(self): return self.\_maxNumbOfStudents  
  
 def get\_dept(self): return self.\_dept  
  
 def get\_instructors(self): return self.\_instructors  
  
 def \_\_str\_\_(self): return self.\_description  
  
 class Time:  
 def \_\_init\_\_(self, block):  
 self.\_block=block  
  
 def get\_block(self): return self.\_block  
  
 def \_\_str\_\_(self): return self.\_block  
  
 class TimePoint:  
 def \_\_init\_\_(self, point, blocks, phantomBlocks):  
 self.\_point=point  
 self.\_blocks=blocks  
 self.\_phantomBlocks=phantomBlocks  
  
 def get\_point(self): return self.\_point  
  
 def get\_blocks(self): return self.\_blocks  
  
 def get\_phantomBlocks(self): return self.\_phantomBlocks  
  
 def \_\_str\_\_(self): return self.\_point  
  
 class Instructor:  
 def \_\_init\_\_(self, number, name, max\_hours, seniority, availability, start, end, overtime):  
 self.\_number=number  
 self.\_name=name  
 self.\_max\_hours=max\_hours  
 self.\_seniority=seniority  
 self.\_availability=availability  
 self.\_start=start  
 self.\_end=end  
 self.\_overtime=overtime  
  
 def get\_number(self): return self.\_number  
  
 def get\_name(self): return self.\_name  
  
 def get\_max\_hours(self): return self.\_max\_hours  
  
 def get\_seniority(self): return self.\_seniority  
  
 def get\_availability(self): return self.\_availability  
  
 def get\_start(self): return self.\_start  
  
 def get\_end(self): return self.\_end  
  
 def get\_overtime(self): return self.\_overtime  
  
 def \_\_str\_\_(self): return self.\_name  
  
 class MeetingTime:  
 def \_\_init\_\_(self, id, time, day, MThour, cell, restTime, breakTime):   
 self.\_id=id  
 self.\_time=time  
 self.\_day=day   
 self.\_MThour=MThour  
 self.\_cell=cell  
 self.\_restTime=restTime   
 self.\_breakTime=breakTime  
  
 def get\_id(self): return self.\_id  
  
 def get\_time(self): return self.\_time  
  
 def get\_day(self): return self.\_day   
  
 def get\_MThour(self): return self.\_MThour  
  
 def get\_cell(self): return self.\_cell  
  
 def get\_restTime(self): return self.\_restTime   
  
 def get\_breakTime(self): return self.\_breakTime  
  
 def \_\_str\_\_(self): return self.\_id  
  
 class MeetingTime1:  
 def \_\_init\_\_(self, id, time, day, MThour, cell, restTime, breakTime):   
 self.\_id=id  
 self.\_time=time  
 self.\_day=day   
 self.\_MThour=MThour  
 self.\_cell=cell  
 self.\_restTime=restTime   
 self.\_breakTime=breakTime  
  
 def get\_id(self): return self.\_id  
  
 def get\_time(self): return self.\_time  
  
 def get\_day(self): return self.\_day   
  
 def get\_MThour(self): return self.\_MThour  
  
 def get\_cell(self): return self.\_cell  
  
 def get\_restTime(self): return self.\_restTime   
  
 def get\_breakTime(self): return self.\_breakTime  
  
 def \_\_str\_\_(self): return self.\_id  
  
 class Room:  
 def \_\_init\_\_(self, number, seatingCapacity, type, dept, availability):   
 self.\_number=number  
 self.\_seatingCapacity=seatingCapacity  
 self.\_type=type   
 self.\_dept=dept  
 self.\_availability=availability  
  
 def get\_number(self): return self.\_number  
  
 def get\_seatingCapacity(self): return self.\_seatingCapacity  
  
 def get\_type(self): return self.\_type   
  
 def get\_dept(self): return self.\_dept  
  
 def get\_availability(self): return self.\_availability  
  
 def \_\_str\_\_(self): return self.\_number  
  
 class Class:  
 def \_\_init\_\_(self, id, section, subject):  
 self.\_id=id  
 self.\_section=section  
 self.\_subject=subject  
 self.\_instructor=None  
 self.\_meetingTime=None  
 self.\_meetingTime1=None  
 self.\_room=None  
  
 def get\_id(self): return self.\_id  
  
 def get\_section(self): return self.\_section  
  
 def get\_subject(self): return self.\_subject  
  
 def get\_instructor(self): return self.\_instructor  
  
 def get\_meetingTime(self): return self.\_meetingTime  
  
 def get\_meetingTime1(self): return self.\_meetingTime1  
  
 def get\_room(self): return self.\_room  
  
 def set\_instructor(self, instructor): self.\_instructor=instructor  
  
 def set\_meetingTime(self, meetingTime): self.\_meetingTime=meetingTime  
  
 def set\_meetingTime1(self, meetingTime1): self.\_meetingTime1=meetingTime1  
  
 def set\_room(self, room): self.\_room=room  
  
 def \_\_str\_\_(self):  
 return str(self.\_section.get\_code()) + **","** + str(self.\_subject.get\_code()) + **","** + \  
 str(self.\_room.get\_number()) + **","** + str(self.\_instructor.get\_number()) + **","** + str(  
 self.\_meetingTime.get\_id()) + **","** + str(self.\_meetingTime1.get\_id())  
  
 class Conflict:  
 class ConflictType(Enum):  
 WRONG\_MTHOUR=1  
 UNEQUAL\_SPLIT=2  
 CASUAL\_SPLITTING=3  
 SAME\_MTS=4  
 OVERLAP\_MTS=5  
  
 NUMB\_OF\_STUDENTS=6  
 ROOM\_AVAILABILITY=7  
 ROOM\_BOOKING=8  
 XDEPT\_ROOM\_UTILITY=9  
  
 INSTRUCTOR\_OVERTIME=10  
 INSTRUCTOR\_AVAILABILITY=11  
 INSTRUCTOR\_BOOKING=12  
 INSTRUCTOR\_OVERLOAD=13  
  
 SECTION\_AVAILABILITY=14  
 SECTION\_BOOKING=15  
  
 LAB\_ON\_VR=16   
 MIXED\_TYPE=17  
  
 def \_\_init\_\_(self, conflictType, conflictBetweenClasses):  
 self.\_conflictType=conflictType  
 self.\_conflictBetweenClasses=conflictBetweenClasses  
  
 def get\_conflictType(self): return self.\_conflictType  
  
 def get\_conflictBetweenClasses(self): return self.\_conflictBetweenClasses  
  
 def \_\_str\_\_(self): return str(self.\_conflictType) + **" "** + str(  
 **" and "**.join(map(str, self.\_conflictBetweenClasses)))  
  
 def click(event):  
 try:  
 selected=tree.focus() *# Grab record position/number* values=tree.item(selected, **'values'**) *# Grab record values* cb\_instructor\_id.delete(0, **'end'**)  
 cb\_meeting\_time.delete(0, **'end'**)  
 cb\_instructor\_id.insert(0, values[0])  
 cb\_meeting\_time.insert(0, values[1])  
 except IndexError: pass  
  
 def combobox\_input0():  
 ARADB()  
 cursor.execute(**"SELECT \* FROM `instructor`"**)  
 conn.commit()  
 data=[]  
 for row in cursor.fetchall():  
 data.append(str(row[0]) + **' ['** + str(row[1]) + **']'**)  
 return data  
 cursor.close()  
 conn.close()  
  
 def combobox\_input1():  
 ARADB()  
 cursor.execute(**"SELECT id FROM `meeting\_time`"**)  
 conn.commit()  
 data=[]  
 for row in cursor.fetchall():  
 data.append(row[0])  
 return data  
 cursor.close()  
 conn.close()  
  
 def combobox\_input2():  
 ARADB()  
 cursor.execute(**"SELECT point FROM `time\_point`"**)  
 conn.commit()  
 data=[]  
 for row in cursor.fetchall():  
 data.append(row[0])  
 return data  
 cursor.close()  
 conn.close()  
  
 def Create(): *# MT INPUTS* if INSTRUCTOR\_ID.get() == **""**:  
 txt\_result.config(text = **"Please choose an instructor!"**, fg = **"red"**)  
 else:  
 dbMgr=DBMgr()  
 meetingTimes=dbMgr.get\_meetingTimes()  
 timepoints=dbMgr.get\_timepoints()  
 sep=**' '** if (not (MON\_START.get() == **""**)):  
 ID\_Mon=list() *# Meeting Time IDs in Monday* Time\_Mon=list()  
 startingTime\_Mon=list()  
 endingTime\_Mon=list()  
 phantom\_time=list()  
 qualifiedMTs\_Mon=list()  
 qualifiedBlocks\_Mon=list()  
 *# List all Mondays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Monday'** == meetingTimes[i].get\_day()):  
 ID\_Mon.append(meetingTimes[i].get\_id())  
 Time\_Mon.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (MON\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Mon.append(timepoints[i].get\_blocks())  
 if (MON\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Mon.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (MON\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Mon)):  
 if (Time\_Mon[i] in startingTime\_Mon[0]):  
 if (Time\_Mon[i] not in endingTime\_Mon[0]):  
 if (Time\_Mon[i] not in phantom\_time[0]):  
 qualifiedMTs\_Mon.append(ID\_Mon[i])  
 qualifiedBlocks\_Mon.append(Time\_Mon[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Mon)):  
 if (Time\_Mon[i] in qualifiedBlocks\_Mon):  
 cursor.execute(  
 **"INSERT INTO 'instructor\_availability' (instructor\_id, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(INSTRUCTOR\_ID.get().split(sep, 1)[0]), str(ID\_Mon[i]), str(MON\_START.get()),  
 str(MON\_END.get())))  
 conn.commit()  
 MON\_START.set(**""**)  
 MON\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 if (not (TUE\_START.get() == **""**)):  
 ID\_Tue=list() *# Meeting Time IDs in Tuesday* Time\_Tue=list()  
 startingTime\_Tue=list()  
 endingTime\_Tue=list()  
 phantom\_time=list()  
 qualifiedMTs\_Tue=list()  
 qualifiedBlocks\_Tue=list()  
 *# List all Tuesdays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Tuesday'** == meetingTimes[i].get\_day()):  
 ID\_Tue.append(meetingTimes[i].get\_id())  
 Time\_Tue.append(meetingTimes[i].get\_time())  
   
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (TUE\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Tue.append(timepoints[i].get\_blocks())  
 if (TUE\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Tue.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (TUE\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Tue)):  
 if (Time\_Tue[i] in startingTime\_Tue[0]):  
 if (Time\_Tue[i] not in endingTime\_Tue[0]):  
 if (Time\_Tue[i] not in phantom\_time[0]):  
 qualifiedMTs\_Tue.append(ID\_Tue[i])  
 qualifiedBlocks\_Tue.append(Time\_Tue[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Tue)):  
 if (Time\_Tue[i] in qualifiedBlocks\_Tue):  
 cursor.execute(  
 **"INSERT INTO 'instructor\_availability' (instructor\_id, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(INSTRUCTOR\_ID.get().split(sep, 1)[0]), str(ID\_Tue[i]), str(TUE\_START.get()),  
 str(TUE\_END.get())))  
 conn.commit()  
 TUE\_START.set(**""**)  
 TUE\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 if (not (WED\_START.get() == **""**)):  
 ID\_Wed=list() *# Meeting Time IDs in Wednesday* Time\_Wed=list()  
 startingTime\_Wed=list()  
 endingTime\_Wed=list()  
 phantom\_time=list()  
 qualifiedMTs\_Wed=list()  
 qualifiedBlocks\_Wed=list()  
 *# List all Wednesdays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Wednesday'** == meetingTimes[i].get\_day()):  
 ID\_Wed.append(meetingTimes[i].get\_id())  
 Time\_Wed.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (WED\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Wed.append(timepoints[i].get\_blocks())  
 if (WED\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Wed.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (WED\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Wed)):  
 if (Time\_Wed[i] in startingTime\_Wed[0]):  
 if (Time\_Wed[i] not in endingTime\_Wed[0]):  
 if (Time\_Wed[i] not in phantom\_time[0]):  
 qualifiedMTs\_Wed.append(ID\_Wed[i])  
 qualifiedBlocks\_Wed.append(Time\_Wed[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Wed)):  
 if (Time\_Wed[i] in qualifiedBlocks\_Wed):  
 cursor.execute(  
 **"INSERT INTO 'instructor\_availability' (instructor\_id, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(INSTRUCTOR\_ID.get().split(sep, 1)[0]), str(ID\_Wed[i]), str(WED\_START.get()),  
 str(WED\_END.get())))  
 conn.commit()  
 WED\_START.set(**""**)  
 WED\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 if (not (THU\_START.get() == **""**)):  
 ID\_Thu=list() *# Meeting Time IDs in Thursday* Time\_Thu=list()  
 startingTime\_Thu=list()  
 endingTime\_Thu=list()  
 phantom\_time=list()  
 qualifiedMTs\_Thu=list()  
 qualifiedBlocks\_Thu=list()  
 *# List all Thursdays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Thursday'** == meetingTimes[i].get\_day()):  
 ID\_Thu.append(meetingTimes[i].get\_id())  
 Time\_Thu.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (THU\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Thu.append(timepoints[i].get\_blocks())  
 if (THU\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Thu.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (THU\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Thu)):  
 if (Time\_Thu[i] in startingTime\_Thu[0]):  
 if (Time\_Thu[i] not in endingTime\_Thu[0]):  
 if (Time\_Thu[i] not in phantom\_time[0]):  
 qualifiedMTs\_Thu.append(ID\_Thu[i])  
 qualifiedBlocks\_Thu.append(Time\_Thu[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Thu)):  
 if (Time\_Thu[i] in qualifiedBlocks\_Thu):  
 cursor.execute(  
 **"INSERT INTO 'instructor\_availability' (instructor\_id, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(INSTRUCTOR\_ID.get().split(sep, 1)[0]), str(ID\_Thu[i]), str(THU\_START.get()),  
 str(THU\_END.get())))  
 conn.commit()  
 THU\_START.set(**""**)  
 THU\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 if (not (FRI\_START.get() == **""**)):  
 ID\_Fri=list() *# Meeting Time IDs in Friday* Time\_Fri=list()  
 startingTime\_Fri=list()  
 endingTime\_Fri=list()  
 phantom\_time=list()  
 qualifiedMTs\_Fri=list()  
 qualifiedBlocks\_Fri=list()  
 *# List all Fridays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Friday'** == meetingTimes[i].get\_day()):  
 ID\_Fri.append(meetingTimes[i].get\_id())  
 Time\_Fri.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (FRI\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Fri.append(timepoints[i].get\_blocks())  
 if (FRI\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Fri.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (FRI\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Fri)):  
 if (Time\_Fri[i] in startingTime\_Fri[0]):  
 if (Time\_Fri[i] not in endingTime\_Fri[0]):  
 if (Time\_Fri[i] not in phantom\_time[0]):  
 qualifiedMTs\_Fri.append(ID\_Fri[i])  
 qualifiedBlocks\_Fri.append(Time\_Fri[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Fri)):  
 if (Time\_Fri[i] in qualifiedBlocks\_Fri):  
 cursor.execute(  
 **"INSERT INTO 'instructor\_availability' (instructor\_id, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(INSTRUCTOR\_ID.get().split(sep, 1)[0]), str(ID\_Fri[i]), str(FRI\_START.get()),  
 str(FRI\_END.get())))  
 conn.commit()  
 FRI\_START.set(**""**)  
 FRI\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 if (not (SAT\_START.get() == **""**)):  
 ID\_Sat=list() *# Meeting Time IDs in Saturday* Time\_Sat=list()  
 startingTime\_Sat=list()  
 endingTime\_Sat=list()  
 phantom\_time=list()  
 qualifiedMTs\_Sat=list()  
 qualifiedBlocks\_Sat=list()  
 *# List all Saturdays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Saturday'** == meetingTimes[i].get\_day()):  
 ID\_Sat.append(meetingTimes[i].get\_id())  
 Time\_Sat.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (SAT\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Sat.append(timepoints[i].get\_blocks())  
 if (SAT\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Sat.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (SAT\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Sat)):  
 if (Time\_Sat[i] in startingTime\_Sat[0]):  
 if (Time\_Sat[i] not in endingTime\_Sat[0]):  
 if (Time\_Sat[i] not in phantom\_time[0]):  
 qualifiedMTs\_Sat.append(ID\_Sat[i])  
 qualifiedBlocks\_Sat.append(Time\_Sat[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Sat)):  
 if (Time\_Sat[i] in qualifiedBlocks\_Sat):  
 cursor.execute(  
 **"INSERT INTO 'instructor\_availability' (instructor\_id, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(INSTRUCTOR\_ID.get().split(sep, 1)[0]), str(ID\_Sat[i]), str(SAT\_START.get()),  
 str(SAT\_END.get())))  
 conn.commit()  
 SAT\_START.set(**""**)  
 SAT\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 if (not (SUN\_START.get() == **""**)):  
 ID\_Sun=list() *# Meeting Time IDs in Sunday* Time\_Sun=list()  
 startingTime\_Sun=list()  
 endingTime\_Sun=list()  
 phantom\_time=list()  
 qualifiedMTs\_Sun=list()  
 qualifiedBlocks\_Sun=list()  
 *# List all Sundays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Sunday'** == meetingTimes[i].get\_day()):  
 ID\_Sun.append(meetingTimes[i].get\_id())  
 Time\_Sun.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (SUN\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Sun.append(timepoints[i].get\_blocks())  
 if (SUN\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Sun.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (SUN\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Sun)):  
 if (Time\_Sun[i] in startingTime\_Sun[0]):  
 if (Time\_Sun[i] not in endingTime\_Sun[0]):  
 if (Time\_Sun[i] not in phantom\_time[0]):  
 qualifiedMTs\_Sun.append(ID\_Sun[i])  
 qualifiedBlocks\_Sun.append(Time\_Sun[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Sun)):  
 if (Time\_Sun[i] in qualifiedBlocks\_Sun):  
 cursor.execute(  
 **"INSERT INTO 'instructor\_availability' (instructor\_id, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(INSTRUCTOR\_ID.get().split(sep, 1)[0]), str(ID\_Sun[i]), str(SUN\_START.get()),  
 str(SUN\_END.get())))  
 conn.commit()  
 SUN\_START.set(**""**)  
 SUN\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Created a data!"**, fg = **"green"**)  
  
 def View():  
 tree.delete(\*tree.get\_children())  
 ARADB()  
 cursor.execute(**"SELECT \* FROM `instructor\_availability` ORDER BY `instructor\_id` ASC"**)  
 fetch=cursor.fetchall()  
 for data in fetch:  
 tree.insert(**''**, **'end'**, values = (data[0], data[1], data[2], data[3]))  
 cursor.close()  
 conn.close()  
 txt\_result.config(text = **"Successfully viewed the data from database"**, fg = **"black"**)  
  
 def Delete():  
 if INSTRUCTOR\_ID.get() == **""**:  
 txt\_result.config(text = **"Please choose an instructor!"**, fg = **"red"**)  
 else:  
 sep=**' '** ARADB()  
 cursor.execute(  
 **"DELETE FROM 'instructor\_availability' WHERE instructor\_id = '"** + str(  
 INSTRUCTOR\_ID.get().split(sep, 1)[0]) + **"'"**)  
 conn.commit()  
 cb\_instructor\_id.delete(0, **'end'**)  
 cb\_meeting\_time.delete(0, **'end'**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Deleted Successfully!"**, fg = **"green"**)  
  
 def Clear():  
 result=messagebox.askquestion(**"Clear"**, **"This will clear the Instructor's Availability table.**\n**Do you want to proceed?"**, icon = **'warning'**)  
 if result == **'yes'**:  
 ARADB()  
 cursor.execute(**"""DROP TABLE instructor\_availability"""**)  
 cursor.execute(**"""create table instructor\_availability (instructor\_id text NOT NULL, meeting\_time\_id text NOT NULL, start text, end text)"""**)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Table Cleared Successfully!"**, fg = **"green"**)  
 else:  
 pass  
  
 def Refresh():  
 IA.destroy()  
 Modify\_Availability()  
  
 *# ==================================VARIABLES==========================================* INSTRUCTOR\_ID=StringVar(IA)  
 MEETING\_TIME=StringVar(IA)  
 MON\_START=StringVar(IA)  
 MON\_END=StringVar(IA)  
 TUE\_START=StringVar(IA)  
 TUE\_END=StringVar(IA)  
 WED\_START=StringVar(IA)  
 WED\_END=StringVar(IA)  
 THU\_START=StringVar(IA)  
 THU\_END=StringVar(IA)  
 FRI\_START=StringVar(IA)  
 FRI\_END=StringVar(IA)  
 SAT\_START=StringVar(IA)  
 SAT\_END=StringVar(IA)  
 SUN\_START=StringVar(IA)  
 SUN\_END=StringVar(IA)  
  
 *# ==================================FRAME==============================================* Top=Frame(IA, width = 900, height = 50, bd = 8, relief = **"raise"**, bg = **"#9EE09E"**)  
 Top.pack(side = TOP)  
 Left=Frame(IA, width = 300, height = 500, bd = 8, relief = **"raise"**)  
 Left.pack(side = LEFT)  
 Right=Frame(IA, width = 600, height = 500, bd = 8, relief = **"raise"**)  
 Right.pack(side = RIGHT)  
 Forms=Frame(Left, width = 300, height = 450)  
 Forms.pack(side = TOP)  
 Buttons=Frame(Left, width = 300, height = 100, bd = 8, relief = **"raise"**)  
 Buttons.pack(side = BOTTOM)  
  
 *# ==================================LABEL WIDGET=======================================* txt\_title=Label(Top, width = 900, font = (**'arial'**, 24), text = **"Instructor Availability"**, fg = **"Black"**,  
 bg = **"#BFE09E"**)   
 txt\_title.pack()  
 txt\_instructor=Label(Forms, text = **"Select Instructor:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_instructor.grid(row = 0, stick = **"e"**)  
  
 txt\_monday=Label(Forms, text = **"Monday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_monday.grid(row = 1, stick = **"e"**)  
 txt\_tuesday=Label(Forms, text = **"Tuesday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_tuesday.grid(row = 2, stick = **"e"**)  
 txt\_wednesday=Label(Forms, text = **"Wednesday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_wednesday.grid(row = 3, stick = **"e"**)  
 txt\_thursday=Label(Forms, text = **"Thursday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_thursday.grid(row = 4, stick = **"e"**)  
 txt\_friday=Label(Forms, text = **"Friday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_friday.grid(row = 5, stick = **"e"**)  
 txt\_saturday=Label(Forms, text = **"Saturday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_saturday.grid(row = 6, stick = **"e"**)  
 txt\_sunday=Label(Forms, text = **"Sunday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_sunday.grid(row = 7, stick = **"e"**)  
 txt\_result=Label(Buttons)  
 txt\_result.pack(side = TOP)  
  
 *# ==================================ENTRY WIDGET=======================================* cb\_instructor\_id=ttk.Combobox(Forms, textvariable = INSTRUCTOR\_ID, width = 44)  
 cb\_instructor\_id[**'values'**]=combobox\_input0()  
 cb\_instructor\_id.place(x = 210, y = 18)  
 cb\_meeting\_time=ttk.Combobox(Forms, textvariable = MEETING\_TIME, width = 20)  
 cb\_meeting\_time[**'values'**]=combobox\_input1()  
  
 txt\_monstart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_monstart.grid(column = 1, row = 1, stick = **"w"**)  
 cb\_monstart=ttk.Combobox(Forms, textvariable = MON\_START, width = 10)  
 cb\_monstart[**'values'**]=combobox\_input2()  
 cb\_monstart.grid(column = 2, row = 1, stick = **"w"**)  
 txt\_monend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 12)  
 txt\_monend.grid(column = 3, row = 1, stick = **"w"**)  
 cb\_monend=ttk.Combobox(Forms, textvariable = MON\_END, width = 10)  
 cb\_monend[**'values'**]=combobox\_input2()  
 cb\_monend.grid(column = 4, row = 1, padx = 10)  
  
 txt\_tuestart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_tuestart.grid(column = 1, row = 2, stick = **"w"**)  
 cb\_tuestart=ttk.Combobox(Forms, textvariable = TUE\_START, width = 10)  
 cb\_tuestart[**'values'**]=combobox\_input2()  
 cb\_tuestart.grid(column = 2, row = 2, stick = **"w"**)  
 txt\_tueend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_tueend.grid(column = 3, row = 2, stick = **"w"**)  
 cb\_tueend=ttk.Combobox(Forms, textvariable = TUE\_END, width = 10)  
 cb\_tueend[**'values'**]=combobox\_input2()  
 cb\_tueend.grid(column = 4, row = 2, padx = 10)  
  
 txt\_wedstart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_wedstart.grid(column = 1, row = 3, stick = **"w"**)  
 cb\_wedstart=ttk.Combobox(Forms, textvariable = WED\_START, width = 10)  
 cb\_wedstart[**'values'**]=combobox\_input2()  
 cb\_wedstart.grid(column = 2, row = 3, stick = **"w"**)  
 txt\_wedend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_wedend.grid(column = 3, row = 3, stick = **"w"**)  
 cb\_wedend=ttk.Combobox(Forms, textvariable = WED\_END, width = 10)  
 cb\_wedend[**'values'**]=combobox\_input2()  
 cb\_wedend.grid(column = 4, row = 3, padx = 10)  
  
 txt\_thustart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_thustart.grid(column = 1, row = 4, stick = **"w"**)  
 cb\_thustart=ttk.Combobox(Forms, textvariable = THU\_START, width = 10)  
 cb\_thustart[**'values'**]=combobox\_input2()  
 cb\_thustart.grid(column = 2, row = 4, stick = **"w"**)  
 txt\_thuend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_thuend.grid(column = 3, row = 4, stick = **"w"**)  
 cb\_thuend=ttk.Combobox(Forms, textvariable = THU\_END, width = 10)  
 cb\_thuend[**'values'**]=combobox\_input2()  
 cb\_thuend.grid(column = 4, row = 4, padx = 10)  
  
 txt\_fristart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_fristart.grid(column = 1, row = 5, stick = **"w"**)  
 cb\_fristart=ttk.Combobox(Forms, textvariable = FRI\_START, width = 10)  
 cb\_fristart[**'values'**]=combobox\_input2()  
 cb\_fristart.grid(column = 2, row = 5, stick = **"w"**)  
 txt\_friend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_friend.grid(column = 3, row = 5, stick = **"w"**)  
 cb\_friend=ttk.Combobox(Forms, textvariable = FRI\_END, width = 10)  
 cb\_friend[**'values'**]=combobox\_input2()  
 cb\_friend.grid(column = 4, row = 5, padx = 10)  
  
 txt\_satstart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_satstart.grid(column = 1, row = 6, stick = **"w"**)  
 cb\_satstart=ttk.Combobox(Forms, textvariable = SAT\_START, width = 10)  
 cb\_satstart[**'values'**]=combobox\_input2()  
 cb\_satstart.grid(column = 2, row = 6, stick = **"w"**)  
 txt\_satend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_satend.grid(column = 3, row = 6, stick = **"w"**)  
 cb\_satend=ttk.Combobox(Forms, textvariable = SAT\_END, width = 10)  
 cb\_satend[**'values'**]=combobox\_input2()  
 cb\_satend.grid(column = 4, row = 6, padx = 10)  
  
 txt\_sunstart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_sunstart.grid(column = 1, row = 7, stick = **"w"**)  
 cb\_sunstart=ttk.Combobox(Forms, textvariable = SUN\_START, width = 10)  
 cb\_sunstart[**'values'**]=combobox\_input2()  
 cb\_sunstart.grid(column = 2, row = 7, stick = **"w"**)  
 txt\_sunend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_sunend.grid(column = 3, row = 7, stick = **"w"**)  
 cb\_sunend=ttk.Combobox(Forms, textvariable = SUN\_END, width = 10)  
 cb\_sunend[**'values'**]=combobox\_input2()  
 cb\_sunend.grid(column = 4, row = 7, padx = 10)  
  
 *# ==================================BUTTONS WIDGET=====================================* btn\_create=Button(Buttons, width = 10, text = **"Create"**, command = Create)  
 btn\_create.pack(side = LEFT)  
 btn\_view=Button(Buttons, width = 10, text = **"View"**, command = View)  
 btn\_view.pack(side = LEFT)  
 btn\_update=Button(Buttons, width = 10, text = **"Update"**, state = DISABLED)  
 btn\_update.pack(side = LEFT)  
 btn\_delete=Button(Buttons, width = 10, text = **"Delete"**, command = Delete)  
 btn\_delete.pack(side = LEFT)  
 btn\_clear=Button(Buttons, width = 10, text = **"Clear"**, fg = **"Black"**, bg = **"#F2B6AE"**, command = Clear)  
 btn\_clear.pack(side = LEFT)  
 btn\_refresh=Button(Top, width = 10, text = **"Refresh (F5)"**, font = (**'arial'**, 10, **'bold'**), fg = **"White"**,  
 bg = **"green"**, command = Refresh)  
 btn\_refresh.pack(side = RIGHT)  
  
 *# ==================================LIST WIDGET========================================* scrollbary=Scrollbar(Right, orient = VERTICAL)  
 scrollbarx=Scrollbar(Right, orient = HORIZONTAL)  
 columns=(**"Instructor"**, **"Meeting Time"**, **"Start"**, **"End"**)  
 tree=ttk.Treeview(Right, columns = columns, selectmode = **"extended"**, height = 500,  
 yscrollcommand = scrollbary.set, xscrollcommand = scrollbarx.set)  
  
 def treeview\_sort\_column(tree, col, reverse):  
 l=[(tree.set(k, col), k) for k in tree.get\_children(**''**)]  
 l.sort(reverse = reverse)  
  
 *# rearrange items in sorted positions* for index, (val, k) in enumerate(l):  
 tree.move(k, **''**, index)  
  
 *# reverse sort next time* tree.heading(col, command = lambda \_col=col: treeview\_sort\_column(tree, \_col, not reverse))  
  
 for col in columns:  
 tree.heading(col, text = col, command = lambda \_col=col: \  
 treeview\_sort\_column(tree, \_col, False))  
 *# for TREE Scroll Bars* scrollbary.config(command = tree.yview)  
 scrollbary.pack(side = RIGHT, fill = Y)  
 scrollbarx.config(command = tree.xview)  
 scrollbarx.pack(side = BOTTOM, fill = X)  
 *# inside the tree view* tree.heading(**'Instructor'**, text = **"Instructor"**, anchor = W)  
 tree.heading(**'Meeting Time'**, text = **"Meeting Time"**, anchor = W)  
 tree.heading(**'Start'**, text = **"Start"**, anchor = W)  
 tree.heading(**'End'**, text = **"End"**, anchor = W)  
 tree.column(**'#0'**, stretch = NO, minwidth = 0, width = 0)  
 tree.column(**'#1'**, stretch = NO, minwidth = 75, width = 75)  
 tree.column(**'#2'**, stretch = NO, minwidth = 150, width = 150)  
 tree.column(**'#3'**, stretch = NO, minwidth = 60, width = 60)  
 tree.bind(**'<ButtonRelease-1>'**, click)  
 tree.pack()  
  
 if \_\_name\_\_ == **'\_\_main\_\_'**:  
 View()  
 IA.mainloop()  
  
 def Modify\_Overtime():  
 IO=Toplevel(instroot)  
 IO.title(**"ECE-SPV"**)  
 IO.wm\_iconbitmap(**"ece-spv.ico"**)  
 IO.screen\_width=IO.winfo\_screenwidth()  
 IO.screen\_height=IO.winfo\_screenheight()  
 IO.width=720  
 IO.height=627  
 IO.x=(screen\_width / 2) - (width / 2)  
 IO.y=(screen\_height / 2) - (height / 2)  
 IO.geometry(**'%dx%d+%d+%d'** % (width, height, x, y))  
 IO.resizable(0, 0)  
  
 *# ==================================COMMANDS============================================* class DBMgr:  
 def \_\_init\_\_(self):  
 self.\_conn=sqlite.connect(**'ARADB.db'**) *# sql connector* self.\_cursor=self.\_conn.cursor() *# sql cursor* self.\_times=self.\_select\_times() *# select time sql cursor* self.\_timepoints=self.\_select\_timepoints() *# select timepoints sql cursor* self.\_meetingTimes=self.\_select\_meeting\_times() *# select meeting time sql cursor* self.\_instructors=self.\_select\_instructors() *# select instructor ql cursor* self.\_subjects=self.\_select\_subjects() *# select subject sql cursor* self.\_rooms=self.\_select\_rooms() *# select room sql cursor* self.\_sections=self.\_select\_sections() *# select section sql cursor* self.\_numberOfClasses=0 *# initial number of classes* for i in range(0, len(self.\_sections)):  
 self.\_numberOfClasses+=len(  
 self.\_sections[i].get\_subjects()) *# auto-itereation of number of classes creation  
  
 # select section sql command* def \_select\_sections(self):  
 self.\_cursor.execute(**"SELECT \* FROM section"**)  
 sections=self.\_cursor.fetchall()  
 returnSections=[]  
 for i in range(0, len(sections)):  
 returnSections.append(Section(sections[i][0], self.\_select\_section\_subjects(sections[i][0]),  
 self.\_select\_section\_availability(sections[i][0])))  
 return returnSections  
  
 *# select subject sql command* def \_select\_subjects(self):  
 self.\_cursor.execute(**"SELECT \* FROM subject"**)  
 subjects=self.\_cursor.fetchall()  
 returnSubjects=[]  
 for i in range(0, len(subjects)):  
 returnSubjects.append(  
 Subject(subjects[i][0], subjects[i][1], subjects[i][2], subjects[i][3], subjects[i][4],  
 subjects[i][5], subjects[i][6], subjects[i][7], self.\_select\_subject\_instructors(  
 subjects[i][0])))   
 return returnSubjects  
  
 *# select instructor sql command* def \_select\_instructors(self):  
 self.\_cursor.execute(**"SELECT \* FROM instructor"**)  
 instructors=self.\_cursor.fetchall()  
 returnInstructors=[]  
 for i in range(0, len(instructors)):  
 returnInstructors.append(  
 Instructor(instructors[i][0], instructors[i][1], instructors[i][2], instructors[i][3],  
 self.\_select\_instructor\_availability(instructors[i][0]),  
 self.\_select\_instructor\_availability\_start(instructors[i][0]),  
 self.\_select\_instructor\_availability\_end(instructors[i][0]),  
 self.\_select\_instructor\_overtime(instructors[i][0])))  
 return returnInstructors  
  
 *# select room sql command* def \_select\_rooms(self):  
 self.\_cursor.execute(**"SELECT \* FROM room"**)  
 rooms=self.\_cursor.fetchall()  
 returnRooms=[]  
 for i in range(0, len(rooms)):  
 returnRooms.append(Room(rooms[i][0], rooms[i][1], rooms[i][2], rooms[i][3],  
 self.\_select\_room\_availability(  
 rooms[i][  
 0])))   
 return returnRooms  
  
 *# select meeting time sql command* def \_select\_meeting\_times(self):  
 self.\_cursor.execute(**"SELECT \* FROM meeting\_time"**)  
 meetingTimes=self.\_cursor.fetchall()  
 returnMeetingTimes=[]  
 for i in range(0, len(meetingTimes)):  
 returnMeetingTimes.append(  
 MeetingTime(meetingTimes[i][0], meetingTimes[i][1], meetingTimes[i][2],  
 meetingTimes[i][3], meetingTimes[i][4],  
 self.\_select\_rest\_time(meetingTimes[i][0]),  
 self.\_select\_break\_time(meetingTimes[i][0])))   
 return returnMeetingTimes  
  
 *# select time sql command* def \_select\_times(self):  
 self.\_cursor.execute(**"SELECT \* FROM time"**)  
 times=self.\_cursor.fetchall()  
 returnTimes=[]  
 for i in range(0, len(times)):  
 returnTimes.append(Time(times[i][0]))  
 return returnTimes  
  
 *# select timepoint sql command* def \_select\_timepoints(self):  
 self.\_cursor.execute(**"SELECT \* FROM time\_point"**)  
 timepoints=self.\_cursor.fetchall()  
 returnTimePoints=[]  
 for i in range(0, len(timepoints)):  
 returnTimePoints.append(TimePoint(timepoints[i][0],  
 self.\_select\_point\_block(timepoints[i][0]),  
 self.\_select\_phantom\_block(timepoints[i][0])))  
 return returnTimePoints  
  
 *# GETTING VALUE* def \_select\_section\_subjects(self, sectionCode):  
 self.\_cursor.execute(**"SELECT \* FROM section\_subject where section\_code == '"** + sectionCode + **"'"**)  
 dbSubjectCodes=self.\_cursor.fetchall()  
 subjectCodes=[]  
 for i in range(0, len(dbSubjectCodes)):  
 subjectCodes.append(dbSubjectCodes[i][1])  
 returnValue=[]  
 for i in range(0, len(self.\_subjects)):  
 if self.\_subjects[i].get\_code() in subjectCodes:  
 returnValue.append(self.\_subjects[i])  
 return returnValue  
  
 def \_select\_subject\_instructors(self, subjectCode):  
 self.\_cursor.execute(**"SELECT \* FROM subject\_instructor where subject\_code == '"** + subjectCode + **"'"**)  
 dbInstructorNumbers=self.\_cursor.fetchall()  
 instructorNumbers=[]  
 for i in range(0, len(dbInstructorNumbers)): instructorNumbers.append(dbInstructorNumbers[i][1])  
 returnValue=[]  
 for i in range(0, len(self.\_instructors)):  
 if self.\_instructors[i].get\_number() in instructorNumbers:  
 returnValue.append(self.\_instructors[i])  
 return returnValue  
  
 def \_select\_instructor\_availability(self, instructor):  
 self.\_cursor.execute(  
 **"SELECT \* from instructor\_availability where instructor\_id == '"** + instructor + **"'"**)  
 instructorMTsRS=self.\_cursor.fetchall()  
 instructorMTs=[]  
 for i in range(0, len(instructorMTsRS)):  
 instructorMTs.append(instructorMTsRS[i][1])  
 instructorAvailability=list()  
 for i in range(0, len(self.\_meetingTimes)):  
 if self.\_meetingTimes[i].get\_id() in instructorMTs:  
 instructorAvailability.append(self.\_meetingTimes[i].get\_id())  
 return instructorAvailability  
  
 def \_select\_instructor\_availability\_start(self, instructor):  
 self.\_cursor.execute(  
 **"SELECT \* from instructor\_availability where instructor\_id == '"** + instructor + **"'"**)  
 instructorMTsRS=self.\_cursor.fetchall()  
 instructorMTsSTART=[]  
 for i in range(0, len(instructorMTsRS)):  
 instructorMTsSTART.append(instructorMTsRS[i][2])  
 instructorAvailabilitySTART=list()  
 for i in range(0, len(self.\_timepoints)):  
 if self.\_timepoints[i].get\_point() in instructorMTsSTART:  
 instructorAvailabilitySTART.append(self.\_timepoints[i].get\_point())  
 return instructorAvailabilitySTART  
  
 def \_select\_instructor\_availability\_end(self, instructor):  
 self.\_cursor.execute(  
 **"SELECT \* from instructor\_availability where instructor\_id == '"** + instructor + **"'"**)  
 instructorMTsRS=self.\_cursor.fetchall()  
 instructorMTsEND=[]  
 for i in range(0, len(instructorMTsRS)):  
 instructorMTsEND.append(instructorMTsRS[i][3])  
 instructorAvailabilityEND=list()  
 for i in range(0, len(self.\_timepoints)):  
 if self.\_timepoints[i].get\_point() in instructorMTsEND:  
 instructorAvailabilityEND.append(self.\_timepoints[i].get\_point())  
 return instructorAvailabilityEND  
  
 def \_select\_instructor\_overtime(self, instructor):  
 self.\_cursor.execute(  
 **"SELECT \* from instructor\_overtime where instructor\_id == '"** + instructor + **"'"**)  
 instructorMTsRS=self.\_cursor.fetchall()  
 instructorMTs=[]  
 for i in range(0, len(instructorMTsRS)):  
 instructorMTs.append(instructorMTsRS[i][1])  
 instructorOvertime=list()  
 for i in range(0, len(self.\_meetingTimes)):  
 if self.\_meetingTimes[i].get\_id() in instructorMTs:  
 instructorOvertime.append(self.\_meetingTimes[i].get\_id())  
 return instructorOvertime  
  
 def \_select\_room\_availability(self, room):  
 self.\_cursor.execute(**"SELECT \* from room\_availability where room\_number = '"** + room + **"'"**)  
 roomMTsRS=self.\_cursor.fetchall()  
 roomMTs=[]  
 for i in range(0, len(roomMTsRS)): roomMTs.append(roomMTsRS[i][1])  
 roomAvailability=list()  
 for i in range(0, len(self.\_meetingTimes)):  
 if self.\_meetingTimes[i].get\_id() in roomMTs:  
 roomAvailability.append(self.\_meetingTimes[i].get\_id())  
 return roomAvailability  
  
 def \_select\_section\_availability(self, section):  
 self.\_cursor.execute(**"SELECT \* from section\_availability where section\_code = '"** + section + **"'"**)  
 sectionMTsRS=self.\_cursor.fetchall()  
 sectionMTs=[]  
 for i in range(0, len(sectionMTsRS)): sectionMTs.append(sectionMTsRS[i][1])  
 sectionAvailability=list()  
 for i in range(0, len(self.\_meetingTimes)):  
 if self.\_meetingTimes[i].get\_id() in sectionMTs:  
 sectionAvailability.append(self.\_meetingTimes[i].get\_id())  
 return sectionAvailability  
  
 def \_select\_rest\_time(self, meeting\_time):  
 self.\_cursor.execute(**"SELECT \* from rest\_time where meeting\_time\_id = '"** + meeting\_time + **"'"**)  
 timeMTsRS=self.\_cursor.fetchall()  
 timeMTs=[]  
 for i in range(0, len(timeMTsRS)): timeMTs.append(timeMTsRS[i][1])  
 restTime=list()  
 for i in range(0, len(self.\_times)):  
 if self.\_times[i].get\_block() in timeMTs:  
 restTime.append(self.\_times[i].get\_block())  
 return restTime  
  
 def \_select\_break\_time(self, meeting\_time):  
 self.\_cursor.execute(**"SELECT \* from break\_time where meeting\_time\_id = '"** + meeting\_time + **"'"**)  
 timeMTsRS=self.\_cursor.fetchall()  
 timeMTs=[]  
 for i in range(0, len(timeMTsRS)): timeMTs.append(timeMTsRS[i][1])  
 breakTime=list()  
 for i in range(0, len(self.\_times)):  
 if self.\_times[i].get\_block() in timeMTs:  
 breakTime.append(self.\_times[i].get\_block())  
 return breakTime  
  
 def \_select\_point\_block(self, point):  
 self.\_cursor.execute(**"SELECT \* from point\_block where time\_point == '"** + point + **"'"**)  
 pointBlocksRS=self.\_cursor.fetchall()  
 pointBlocks=[]  
 for i in range(0, len(pointBlocksRS)): pointBlocks.append(pointBlocksRS[i][1])  
 returnValue=list()  
 for i in range(0, len(self.\_times)):  
 if self.\_times[i].get\_block() in pointBlocks:  
 returnValue.append(self.\_times[i].get\_block())  
 return returnValue  
  
 def \_select\_phantom\_block(self, point):  
 self.\_cursor.execute(**"SELECT \* from phantom\_block where time\_point == '"** + point + **"'"**)  
 pointBlocksRS=self.\_cursor.fetchall()  
 pointBlocks=[]  
 for i in range(0, len(pointBlocksRS)): pointBlocks.append(pointBlocksRS[i][1])  
 returnValue=list()  
 for i in range(0, len(self.\_times)):  
 if self.\_times[i].get\_block() in pointBlocks:  
 returnValue.append(self.\_times[i].get\_block())  
 return returnValue  
  
 def get\_sections(self):  
 return self.\_sections *# get item to section table* def get\_subjects(self):  
 return self.\_subjects *# get item to subject table* def get\_instructors(self):  
 return self.\_instructors *# get item to instructor table* def get\_meetingTimes(self):  
 return self.\_meetingTimes *# get item to meeting time table* def get\_times(self):  
 return self.\_times *# get item to time table* def get\_timepoints(self):  
 return self.\_timepoints *# get item to time table* def get\_rooms(self):  
 return self.\_rooms *# get item to room table* def get\_numberOfClasses(self):  
 return self.\_numberOfClasses *# get item to number of class table* class Schedule:  
 def \_\_init\_\_(self):  
 self.\_data=dbMgr  
 self.\_classes=[]  
 self.\_conflicts=[]  
 self.\_fitness=-1  
 self.\_classNumb=0  
 self.\_isFitnessChanged=True  
 self.\_generationNumber=0  
  
 def get\_classes(self):  
 self.\_isFitnessChanged=True  
 return self.\_classes  
  
 def get\_conflicts(self):  
 return self.\_conflicts  
  
 def get\_fitness(self):  
 if (self.\_isFitnessChanged == True):  
 self.\_fitness=self.calculate\_fitness()  
 self.\_isFitnessChanged=False  
 return self.\_fitness  
  
 def initialize(self):  
 sections=dbMgr.get\_sections() *# point to section table* meetingTimes=dbMgr.get\_meetingTimes()  
 rooms=dbMgr.get\_rooms()  
 classes=self.\_classes  
 PR=list()  
 instructorList=list()  
 MT1=list()  
 MT1p5=list()  
 MT2=list()  
 MT3=list()  
 MT4=list()  
 MT1v2=list()  
 MT1p5v2=list()  
 MT2v2=list()  
 MT3v2=list()  
 for RM in range(0, len(rooms)):  
 if (rooms[RM].get\_type() == **'PR'**):  
 PR.append(rooms[RM])  
 for MT in range(0, len(meetingTimes)):  
 if (meetingTimes[MT].get\_MThour() == 1):  
 MT1.append(meetingTimes[MT])  
 MT1v2.append(meetingTimes[MT])  
 if (meetingTimes[MT].get\_MThour() == 1.5):  
 MT1p5.append(meetingTimes[MT])  
 MT1p5v2.append(meetingTimes[MT])  
 if (meetingTimes[MT].get\_MThour() == 2):  
 MT2.append(meetingTimes[MT])  
 MT2v2.append(meetingTimes[MT])  
 if (meetingTimes[MT].get\_MThour() == 3):  
 MT3.append(meetingTimes[MT])  
 MT3v2.append(meetingTimes[MT])  
 if (meetingTimes[MT].get\_MThour() == 4):  
 MT4.append(meetingTimes[MT])  
 for i in range(0, len(sections)): *# List all the items per coloumn* subjects=sections[i].get\_subjects() *# point to subject table* for j in range(0, len(subjects)): *# List all the items per coloumn* newClass=Class(self.\_classNumb, sections[i], subjects[j]) *# Create a new class* self.\_classNumb+=1 *# Iterate the class creation* newClass.set\_instructor(subjects[j].get\_instructors()[rnd.randrange(0, len(subjects[  
 j].get\_instructors()))]) *# Pick a random instructor taht can teach the assigned subject for the new class* if (subjects[j].get\_compatibility() == **'PR'**):  
 newClass.set\_room(  
 PR[rnd.randrange(0, len(PR))]) *# Pick a random room in database for the new class* else:  
 newClass.set\_room(  
 rooms[rnd.randrange(0,  
 len(rooms))]) *# Pick a random room in database for the new class"""* def wholetwohours():  
 newClass.set\_meetingTime(MT2[rnd.randrange(0, len(MT2))])  
 newClass.set\_meetingTime1(dbMgr.get\_meetingTimes()[0])  
  
 def wholethreehours():  
 newClass.set\_meetingTime(MT3[rnd.randrange(0, len(MT3))])  
 newClass.set\_meetingTime1(dbMgr.get\_meetingTimes()[0])  
  
 def wholefourhours():  
 newClass.set\_meetingTime(MT4[rnd.randrange(0, len(MT4))])  
 newClass.set\_meetingTime1(dbMgr.get\_meetingTimes()[0])  
  
 def splittwohours():  
 newClass.set\_meetingTime(MT1[rnd.randrange(0, len(MT1))])  
 newClass.set\_meetingTime1(MT1v2[rnd.randrange(0, len(MT1v2))])  
  
 def splitthreehours():  
 newClass.set\_meetingTime(MT1p5[rnd.randrange(0, len(MT1p5))])  
 newClass.set\_meetingTime1(MT1p5v2[rnd.randrange(0, len(MT1p5v2))])  
  
 def splitfourhours():  
 newClass.set\_meetingTime(MT2[rnd.randrange(0, len(MT2))])  
 newClass.set\_meetingTime1(MT2v2[rnd.randrange(0, len(MT2v2))])  
  
 def call\_funcs\_randomly(funcs):  
 shuffle(funcs)  
 for func in funcs:  
 func()  
  
 *# with mixed Split Time* if (subjects[j].get\_numbHour() == 1):  
 newClass.set\_meetingTime(MT1[rnd.randrange(0, len(MT1))])  
 newClass.set\_meetingTime1(dbMgr.get\_meetingTimes()[0])  
 elif (subjects[j].get\_numbHour() == 2):  
 call\_funcs\_randomly([wholetwohours, splittwohours])  
 elif (subjects[j].get\_numbHour() == 3):  
 call\_funcs\_randomly([wholethreehours, splitthreehours])  
 elif (subjects[j].get\_numbHour() == 4):  
 call\_funcs\_randomly([wholefourhours, splitfourhours])  
 elif (subjects[j].get\_numbHour() == 6):  
 newClass.set\_meetingTime(MT3[rnd.randrange(0, len(MT3))])  
 newClass.set\_meetingTime1(MT3v2[rnd.randrange(0, len(MT3v2))])  
 else:  
 newClass.set\_meetingTime(  
 dbMgr.get\_meetingTimes()[rnd.randrange(0, len(dbMgr.get\_meetingTimes()))])  
 newClass.set\_meetingTime1(  
 dbMgr.get\_meetingTimes()[rnd.randrange(0, len(dbMgr.get\_meetingTimes()))])  
 self.\_classes.append(newClass) *# Add result as new class* return self  
  
 def calculate\_fitness(self):  
 self.\_conflicts=[]  
 classes=self.get\_classes()  
 instructorNames=list()  
 instructor\_Hours=list()  
 for i in range(0, len(classes)):  
 instructorNames.append(classes[i].get\_instructor().get\_name())  
 subjectHours=list()  
 if (classes[i].get\_meetingTime1().get\_id() != **'NULL'**):  
 if (classes[i].get\_meetingTime().get\_id() == classes[i].get\_meetingTime1().get\_id()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(Conflict(Conflict.ConflictType.SAME\_MTS, conflictBetweenClasses))  
 if DISABLE\_UNEQUAL\_SPLIT == True:  
 if (classes[i].get\_meetingTime().get\_MThour() != classes[  
 i].get\_meetingTime1().get\_MThour()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.UNEQUAL\_SPLIT, conflictBetweenClasses))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 i].get\_meetingTime1().get\_restTime()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.OVERLAP\_MTS, conflictBetweenClasses))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 i].get\_meetingTime().get\_restTime()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.OVERLAP\_MTS, conflictBetweenClasses))  
 if DISABLE\_CASUAL\_SPLITTING == True:  
 *# Disable Same Day Split Time* if (classes[i].get\_meetingTime().get\_day() == classes[i].get\_meetingTime1().get\_day()):  
 if (classes[i].get\_meetingTime().get\_MThour() != classes[  
 i].get\_subject().get\_numbHour()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.CASUAL\_SPLITTING, conflictBetweenClasses))  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[i].get\_meetingTime1().get\_MThour()) !=  
 classes[i].get\_subject().get\_numbHour()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.WRONG\_MTHOUR, conflictBetweenClasses))  
 if ENABLE\_NUMB\_OF\_STUDENTS == True:  
 if (classes[i].get\_room().get\_seatingCapacity() < classes[  
 i].get\_subject().get\_maxNumbOfStudents()):  
 seatingCapacityConflict=list()  
 seatingCapacityConflict.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.NUMB\_OF\_STUDENTS, seatingCapacityConflict))  
 *# Disable to assign Labs on VR* if LAB\_ON\_VR == False:  
 if (classes[i].get\_subject().get\_compatibility() == **'PR'**):  
 if (classes[i].get\_room().get\_type() != **'PR'**):  
 roomCompatibilityConflict=list()  
 roomCompatibilityConflict.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.LAB\_ON\_VR, roomCompatibilityConflict))  
 *# Disable cross department room utility* if DISABLE\_XDEPT\_ROOM\_UTILITY == True:  
 if (classes[i].get\_room().get\_type() != **'VR'**):  
 if (classes[i].get\_subject().get\_dept() != classes[i].get\_room().get\_dept()):  
 roomUtilityConflict=list()  
 roomUtilityConflict.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.XDEPT\_ROOM\_UTILITY, roomUtilityConflict))  
  
 *# if ENABLE\_INSTRUCTOR\_OVERTIME == True:* if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 0* if (classes[i].get\_meetingTime().get\_id() in classes[  
 i].get\_instructor().get\_overtime()): *# This identify the schedules inside the overtime* if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_OVERTIME, conflictBetweenClasses))  
 else:  
 *# 0* if (classes[i].get\_meetingTime().get\_id() in classes[i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_instructor().get\_availability()): *# This removes the schedules out side the official time and overtime* conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_OVERTIME, conflictBetweenClasses))  
 *# 1* if (classes[i].get\_meetingTime1().get\_id() in classes[i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime1().get\_id() not in classes[  
 i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_OVERTIME, conflictBetweenClasses))  
 *# Limit Instructor Availibilty* if LIMIT\_INSTRUCTOR\_AVAILABILITY == True:  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_AVAILABILITY,  
 conflictBetweenClasses))  
 else:  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_AVAILABILITY,  
 conflictBetweenClasses))  
 *# 1* if (classes[i].get\_meetingTime1().get\_id() not in classes[  
 i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime1().get\_id() not in classes[  
 i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_AVAILABILITY,  
 conflictBetweenClasses))  
 *# Limit Room Availibilty* if LIMIT\_ROOM\_AVAILABILITY == True:  
 if (classes[i].get\_room().get\_type() != **'VR'**):  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_room().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_AVAILABILITY, conflictBetweenClasses))  
 else:  
 *# 1* if (classes[i].get\_meetingTime1().get\_id() not in classes[  
 i].get\_room().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_AVAILABILITY, conflictBetweenClasses))  
 *# Limit Section Availibilty* if LIMIT\_SECTION\_AVAILABILITY == True:  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_section().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_AVAILABILITY, conflictBetweenClasses))  
 else:  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[  
 i].get\_section().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_AVAILABILITY, conflictBetweenClasses))  
 *# 1* if (classes[i].get\_meetingTime1().get\_id() not in classes[  
 i].get\_section().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_AVAILABILITY, conflictBetweenClasses))  
  
 for j in range(0, len(classes)):  
 *# Class Optimization* if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day() and  
 classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[  
 j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 else:  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day() and  
 classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[  
 j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 *# 11* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime1().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime1().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[  
 j].get\_meetingTime1().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime1().get\_breakTime()):  
 if ((classes[i].get\_meetingTime1().get\_MThour() + classes[  
 j].get\_meetingTime1().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 *# 10* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime1().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime1().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime1().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime1().get\_MThour() + classes[  
 j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 *# 01* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime1().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[  
 j].get\_meetingTime1().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 instructorBookingConflict))  
 *# Instructor Optimization* if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[  
 j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 else:  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[  
 j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# 11* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime1().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime1().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[  
 j].get\_meetingTime1().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime1().get\_breakTime()):  
 if ((classes[i].get\_meetingTime1().get\_MThour() + classes[  
 j].get\_meetingTime1().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# 01* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime1().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime1().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime1().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime1().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[  
 j].get\_meetingTime1().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# 10* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime1().get\_MThour() + classes[  
 j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# Room Optimization* if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day() and  
 classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 else:  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day() and  
 classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 *# 11* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime1().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime1().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[  
 j].get\_meetingTime1().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 *# 01* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime1().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[  
 j].get\_meetingTime1().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 if (classes[i].get\_meetingTime().get\_time() == classes[  
 j].get\_meetingTime1().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 *# 10* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime().get\_day()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[  
 j].get\_meetingTime().get\_restTime()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[  
 j].get\_meetingTime().get\_time()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, instructorBookingConflict))  
 if (j >= i):  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_meetingTime().get\_id() == classes[j].get\_meetingTime().get\_id()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, roomBookingConflict))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 else:  
 *# 00* if (classes[i].get\_meetingTime().get\_id() == classes[j].get\_meetingTime().get\_id()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, roomBookingConflict))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# 11* if (classes[i].get\_meetingTime1().get\_id() == classes[j].get\_meetingTime1().get\_id()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, roomBookingConflict))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# 01* if (classes[i].get\_meetingTime().get\_id() == classes[j].get\_meetingTime1().get\_id()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, roomBookingConflict))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# 10* if (classes[i].get\_meetingTime1().get\_id() == classes[j].get\_meetingTime().get\_id()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, roomBookingConflict))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 instructorBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 instructorBookingConflict))  
 *# Disable Dual Class Type* if DISABLE\_MIXED\_TYPE == True:  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime().get\_day() == classes[  
 j].get\_meetingTime().get\_day()):   
 if (classes[i].get\_room().get\_type() != classes[  
 j].get\_room().get\_type()):   
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 roomBookingConflict)) *# Room Type Conflict* else:  
 *# 00* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime().get\_day() == classes[  
 j].get\_meetingTime().get\_day()):   
 if (classes[i].get\_room().get\_type() != classes[  
 j].get\_room().get\_type()):   
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 roomBookingConflict)) *# Room Type Conflict  
 # 11* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime1().get\_day() == classes[  
 j].get\_meetingTime1().get\_day()):   
 if (classes[i].get\_room().get\_type() != classes[  
 j].get\_room().get\_type()):   
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 roomBookingConflict)) *# Room Type Conflict  
 # 01* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime().get\_day() == classes[  
 j].get\_meetingTime1().get\_day()):   
 if (classes[i].get\_room().get\_type() != classes[  
 j].get\_room().get\_type()):   
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 roomBookingConflict)) *# Room Type Conflict  
 # 10* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()  
 and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime1().get\_day() == classes[  
 j].get\_meetingTime().get\_day()):   
 if (classes[i].get\_room().get\_type() != classes[  
 j].get\_room().get\_type()):   
 roomBookingConflict=list()  
 roomBookingConflict.append(classes[i])  
 roomBookingConflict.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 roomBookingConflict)) *# Room Type Conflict* if (classes[j].get\_instructor().get\_name() == instructorNames[i]):  
 subjectHours.append(float(classes[j].get\_subject().get\_numbHour()))  
 *# Enable Unit Limit* instructor\_Hours.append(subjectHours)  
 if DISABLE\_INSTRUCTOR\_OVERLOAD == True:  
 if (classes[i].get\_instructor().get\_max\_hours() < sum(instructor\_Hours[i])):  
 instructorBookingConflict=list()  
 instructorBookingConflict.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_OVERLOAD, instructorBookingConflict))  
 return 1 / ((1.0 \* len(self.\_conflicts) + 1))  
  
 def \_\_str\_\_(self):  
 returnValue=**""** for i in range(0, len(self.\_classes) - 1):  
 returnValue+=str(self.\_classes[i]) + **", "** returnValue+=str(self.\_classes[len(self.\_classes) - 1])  
 return returnValue  
  
 class Population:  
 def \_\_init\_\_(self, size):  
 self.\_size=size  
 self.\_data=dbMgr  
 self.\_schedules=[]  
 for i in range(0, size): self.\_schedules.append(Schedule().initialize())  
  
 def get\_size(self): return self.\_size  
  
 def get\_schedules(self): return self.\_schedules  
  
 class GeneticAlgorithm:  
 def evolve(self, population):  
 return self.\_mutate\_population(self.\_crossover\_population(population))  
  
 def \_crossover\_population(self, pop):  
 crossover\_pop=Population(0)  
 for i in range(NUMB\_OF\_ELITE\_SCHEDULES):  
 crossover\_pop.get\_schedules().append(pop.get\_schedules()[i])  
 i=NUMB\_OF\_ELITE\_SCHEDULES  
 while i < POPULATION\_SIZE:  
 schedule1=self.\_select\_tournament\_population(pop).get\_schedules()[0]  
 schedule2=self.\_select\_tournament\_population(pop).get\_schedules()[0]  
 crossover\_pop.get\_schedules().append(self.\_crossover\_schedule(schedule1, schedule2))  
 i+=1  
 return crossover\_pop  
  
 def \_mutate\_population(self, population):  
 for i in range(NUMB\_OF\_ELITE\_SCHEDULES, POPULATION\_SIZE):  
 self.\_mutate\_schedule(population.get\_schedules()[i])  
 return population  
  
 def \_crossover\_schedule(self, schedule1, schedule2):  
 crossoverSchedule=Schedule().initialize()  
 for i in range(0, len(crossoverSchedule.get\_classes())):  
 if (rnd.random() > 0.5):  
 crossoverSchedule.get\_classes()[i]=schedule1.get\_classes()[i]  
 else:  
 crossoverSchedule.get\_classes()[i]=schedule2.get\_classes()[i]  
 return crossoverSchedule  
  
 def \_mutate\_schedule(self, mutateSchedule):  
 schedule=Schedule().initialize()  
 for i in range(0, len(mutateSchedule.get\_classes())):  
 if (MUTATION\_RATE > rnd.random()): mutateSchedule.get\_classes()[i]=schedule.get\_classes()[i]  
 return mutateSchedule  
  
 def \_select\_tournament\_population(self, pop):  
 tournament\_pop=Population(0)  
 i=0  
 while i < TOURNAMENT\_SELECTION\_SIZE:  
 tournament\_pop.get\_schedules().append(  
 pop.get\_schedules()[rnd.randrange(0, int(POPULATION\_SIZE))])  
 i+=1  
 tournament\_pop.get\_schedules().sort(key = lambda x: x.get\_fitness(), reverse = True)  
 return tournament\_pop  
  
 class Section:  
 def \_\_init\_\_(self, code, subjects, availability):  
 self.\_code=code  
 self.\_subjects=subjects  
 self.\_availability=availability  
  
 def get\_code(self): return self.\_code  
  
 def get\_subjects(self): return self.\_subjects  
  
 def get\_availability(self): return self.\_availability  
  
 class Subject:  
 def \_\_init\_\_(self, code, description, units, numbHour, compatibility, curriculum, maxNumbOfStudents,  
 dept,  
 instructors):  
 self.\_code=code  
 self.\_description=description  
 self.\_units=units  
 self.\_numbHour=numbHour  
 self.\_compatibility=compatibility  
 self.\_curriculum=curriculum  
 self.\_maxNumbOfStudents=maxNumbOfStudents  
 self.\_dept=dept  
 self.\_instructors=instructors  
  
 def get\_code(self): return self.\_code  
  
 def get\_description(self): return self.\_description  
  
 def get\_units(self): return self.\_units  
  
 def get\_numbHour(self): return self.\_numbHour  
  
 def get\_compatibility(self): return self.\_compatibility  
  
 def get\_curriculum(self): return self.\_curriculum  
  
 def get\_maxNumbOfStudents(self): return self.\_maxNumbOfStudents  
  
 def get\_dept(self): return self.\_dept  
  
 def get\_instructors(self): return self.\_instructors  
  
 def \_\_str\_\_(self): return self.\_description  
  
 class Time:  
 def \_\_init\_\_(self, block):  
 self.\_block=block  
  
 def get\_block(self): return self.\_block  
  
 def \_\_str\_\_(self): return self.\_block  
  
 class TimePoint:  
 def \_\_init\_\_(self, point, blocks, phantomBlocks):  
 self.\_point=point  
 self.\_blocks=blocks  
 self.\_phantomBlocks=phantomBlocks  
  
 def get\_point(self): return self.\_point  
  
 def get\_blocks(self): return self.\_blocks  
  
 def get\_phantomBlocks(self): return self.\_phantomBlocks  
  
 def \_\_str\_\_(self): return self.\_point  
  
 class Instructor:  
 def \_\_init\_\_(self, number, name, max\_hours, seniority, availability, start, end, overtime):  
 self.\_number=number  
 self.\_name=name  
 self.\_max\_hours=max\_hours  
 self.\_seniority=seniority  
 self.\_availability=availability  
 self.\_start=start  
 self.\_end=end  
 self.\_overtime=overtime  
  
 def get\_number(self): return self.\_number  
  
 def get\_name(self): return self.\_name  
  
 def get\_max\_hours(self): return self.\_max\_hours  
  
 def get\_seniority(self): return self.\_seniority  
  
 def get\_availability(self): return self.\_availability  
  
 def get\_start(self): return self.\_start  
  
 def get\_end(self): return self.\_end  
  
 def get\_overtime(self): return self.\_overtime  
  
 def \_\_str\_\_(self): return self.\_name  
  
 class MeetingTime:  
 def \_\_init\_\_(self, id, time, day, MThour, cell, restTime, breakTime):   
 self.\_id=id  
 self.\_time=time  
 self.\_day=day   
 self.\_MThour=MThour  
 self.\_cell=cell  
 self.\_restTime=restTime   
 self.\_breakTime=breakTime  
  
 def get\_id(self): return self.\_id  
  
 def get\_time(self): return self.\_time  
  
 def get\_day(self): return self.\_day   
  
 def get\_MThour(self): return self.\_MThour  
  
 def get\_cell(self): return self.\_cell  
  
 def get\_restTime(self): return self.\_restTime   
  
 def get\_breakTime(self): return self.\_breakTime  
  
 def \_\_str\_\_(self): return self.\_id  
  
 class MeetingTime1:  
 def \_\_init\_\_(self, id, time, day, MThour, cell, restTime, breakTime):   
 self.\_id=id  
 self.\_time=time  
 self.\_day=day   
 self.\_MThour=MThour  
 self.\_cell=cell  
 self.\_restTime=restTime   
 self.\_breakTime=breakTime  
  
 def get\_id(self): return self.\_id  
  
 def get\_time(self): return self.\_time  
  
 def get\_day(self): return self.\_day   
  
 def get\_MThour(self): return self.\_MThour  
  
 def get\_cell(self): return self.\_cell  
  
 def get\_restTime(self): return self.\_restTime   
  
 def get\_breakTime(self): return self.\_breakTime  
  
 def \_\_str\_\_(self): return self.\_id  
  
 class Room:  
 def \_\_init\_\_(self, number, seatingCapacity, type, dept, availability):   
 self.\_number=number  
 self.\_seatingCapacity=seatingCapacity  
 self.\_type=type   
 self.\_dept=dept  
 self.\_availability=availability  
  
 def get\_number(self): return self.\_number  
  
 def get\_seatingCapacity(self): return self.\_seatingCapacity  
  
 def get\_type(self): return self.\_type   
  
 def get\_dept(self): return self.\_dept  
  
 def get\_availability(self): return self.\_availability  
  
 def \_\_str\_\_(self): return self.\_number  
  
 class Class:  
 def \_\_init\_\_(self, id, section, subject):  
 self.\_id=id  
 self.\_section=section  
 self.\_subject=subject  
 self.\_instructor=None  
 self.\_meetingTime=None  
 self.\_meetingTime1=None  
 self.\_room=None  
  
 def get\_id(self): return self.\_id  
  
 def get\_section(self): return self.\_section  
  
 def get\_subject(self): return self.\_subject  
  
 def get\_instructor(self): return self.\_instructor  
  
 def get\_meetingTime(self): return self.\_meetingTime  
  
 def get\_meetingTime1(self): return self.\_meetingTime1  
  
 def get\_room(self): return self.\_room  
  
 def set\_instructor(self, instructor): self.\_instructor=instructor  
  
 def set\_meetingTime(self, meetingTime): self.\_meetingTime=meetingTime  
  
 def set\_meetingTime1(self, meetingTime1): self.\_meetingTime1=meetingTime1  
  
 def set\_room(self, room): self.\_room=room  
  
 def \_\_str\_\_(self):  
 return str(self.\_section.get\_code()) + **","** + str(self.\_subject.get\_code()) + **","** + \  
 str(self.\_room.get\_number()) + **","** + str(self.\_instructor.get\_number()) + **","** + str(  
 self.\_meetingTime.get\_id()) + **","** + str(self.\_meetingTime1.get\_id())  
  
 class Conflict:  
 class ConflictType(Enum):  
 WRONG\_MTHOUR=1  
 UNEQUAL\_SPLIT=2  
 CASUAL\_SPLITTING=3  
 SAME\_MTS=4  
 OVERLAP\_MTS=5  
  
 NUMB\_OF\_STUDENTS=6  
 ROOM\_AVAILABILITY=7  
 ROOM\_BOOKING=8  
 XDEPT\_ROOM\_UTILITY=9  
  
 INSTRUCTOR\_OVERTIME=10  
 INSTRUCTOR\_AVAILABILITY=11  
 INSTRUCTOR\_BOOKING=12  
 INSTRUCTOR\_OVERLOAD=13  
  
 SECTION\_AVAILABILITY=14  
 SECTION\_BOOKING=15  
  
 LAB\_ON\_VR=16   
 MIXED\_TYPE=17  
  
 def \_\_init\_\_(self, conflictType, conflictBetweenClasses):  
 self.\_conflictType=conflictType  
 self.\_conflictBetweenClasses=conflictBetweenClasses  
  
 def get\_conflictType(self): return self.\_conflictType  
  
 def get\_conflictBetweenClasses(self): return self.\_conflictBetweenClasses  
  
 def \_\_str\_\_(self): return str(self.\_conflictType) + **" "** + str(  
 **" and "**.join(map(str, self.\_conflictBetweenClasses)))  
  
 def click(event):  
 try:  
 selected=tree.focus() *# Grab record position/number* values=tree.item(selected, **'values'**) *# Grab record values* cb\_instructor\_id.delete(0, **'end'**)  
 cb\_meeting\_time.delete(0, **'end'**)  
 cb\_instructor\_id.insert(0, values[0])  
 cb\_meeting\_time.insert(0, values[1])  
 except IndexError: pass  
  
 def combobox\_input0():  
 ARADB()  
 cursor.execute(**"SELECT \* FROM `instructor`"**)  
 conn.commit()  
 data=[]  
 for row in cursor.fetchall():  
 data.append(str(row[0]) + **' ['** + str(row[1]) + **']'**)  
 return data  
 cursor.close()  
 conn.close()  
  
 def combobox\_input1():  
 ARADB()  
 cursor.execute(**"SELECT id FROM `meeting\_time`"**)  
 conn.commit()  
 data=[]  
 for row in cursor.fetchall():  
 data.append(row[0])  
 return data  
 cursor.close()  
 conn.close()  
  
 def combobox\_input2():  
 ARADB()  
 cursor.execute(**"SELECT point FROM `time\_point`"**)  
 conn.commit()  
 data=[]  
 for row in cursor.fetchall():  
 data.append(row[0])  
 return data  
 cursor.close()  
 conn.close()  
  
 def Create(): *# MT INPUTS* if INSTRUCTOR\_ID.get() == **""**:  
 txt\_result.config(text = **"Please choose an instructor!"**, fg = **"red"**)  
 else:  
 dbMgr=DBMgr()  
 meetingTimes=dbMgr.get\_meetingTimes()  
 timepoints=dbMgr.get\_timepoints()  
 sep=**' '** instructors=dbMgr.get\_instructors()  
 Instructor = []  
 Day = []  
 MonStart = []  
 TueStart = []  
 WedStart = []  
 ThuStart = []  
 FriStart = []  
 SatStart = []  
 SunStart = []  
 for i in range(0, len(instructors)):  
 Instructor.append(instructors[i])  
 day = [**'Monday'**, **'Tuesday'**, **'Wednesday'**, **'Thursday'**, **'Friday'**, **'Saturday'**, **'Sunday'**]  
 monStart = []  
 tueStart = []  
 wedStart = []  
 thuStart = []  
 friStart = []  
 satStart = []  
 sunStart = []  
 for j in range(0, len(meetingTimes)):  
 if meetingTimes[j].get\_id() in instructors[i].get\_availability():  
 for k in range(0, len(timepoints)):  
 if meetingTimes[j].get\_day() == day[0]:  
 if (re.split(**"-"**, meetingTimes[j].get\_time())[0]) == timepoints[k].get\_point():  
 if timepoints[k].get\_point() not in monStart:  
 monStart.append(timepoints[k].get\_point())  
 if meetingTimes[j].get\_day() == day[1]:  
 if (re.split(**"-"**, meetingTimes[j].get\_time())[0]) == timepoints[k].get\_point():  
 if timepoints[k].get\_point() not in tueStart:  
 tueStart.append(timepoints[k].get\_point())  
 if meetingTimes[j].get\_day() == day[2]:  
 if (re.split(**"-"**, meetingTimes[j].get\_time())[0]) == timepoints[k].get\_point():  
 if timepoints[k].get\_point() not in wedStart:  
 wedStart.append(timepoints[k].get\_point())  
 if meetingTimes[j].get\_day() == day[3]:  
 if (re.split(**"-"**, meetingTimes[j].get\_time())[0]) == timepoints[k].get\_point():  
 if timepoints[k].get\_point() not in thuStart:  
 thuStart.append(timepoints[k].get\_point())  
 if meetingTimes[j].get\_day() == day[4]:  
 if (re.split(**"-"**, meetingTimes[j].get\_time())[0]) == timepoints[k].get\_point():  
 if timepoints[k].get\_point() not in friStart:  
 friStart.append(timepoints[k].get\_point())  
 if meetingTimes[j].get\_day() == day[5]:  
 if (re.split(**"-"**, meetingTimes[j].get\_time())[0]) == timepoints[k].get\_point():  
 if timepoints[k].get\_point() not in satStart:  
 satStart.append(timepoints[k].get\_point())  
 if meetingTimes[j].get\_day() == day[6]:  
 if (re.split(**"-"**, meetingTimes[j].get\_time())[0]) == timepoints[k].get\_point():  
 if timepoints[k].get\_point() not in sunStart:  
 sunStart.append(timepoints[k].get\_point())  
 if not monStart: monStart.append(**'NULL'**)  
 if not tueStart: tueStart.append(**'NULL'**)  
 if not wedStart: wedStart.append(**'NULL'**)  
 if not thuStart: thuStart.append(**'NULL'**)  
 if not friStart: friStart.append(**'NULL'**)  
 if not satStart: satStart.append(**'NULL'**)  
 if not sunStart: sunStart.append(**'NULL'**)  
 MonStart.append(monStart)  
 TueStart.append(tueStart)  
 WedStart.append(wedStart)  
 ThuStart.append(thuStart)  
 FriStart.append(friStart)  
 SatStart.append(satStart)  
 SunStart.append(sunStart)  
 Day.append(day)  
 if (not (MON\_START.get() == **""**)):  
 ID\_Mon=list() *# Meeting Time IDs in Monday* Time\_Mon=list()  
 startingTime\_Mon=list()  
 endingTime\_Mon=list()  
 over\_phantom\_time=list()  
 phantom\_time=list()  
 qualifiedMTs\_Mon=list()  
 qualifiedBlocks\_Mon=list()  
 *# List all Mondays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Monday'** == meetingTimes[i].get\_day()):  
 ID\_Mon.append(meetingTimes[i].get\_id())  
 Time\_Mon.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (MON\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Mon.append(timepoints[i].get\_blocks())  
 if (MON\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Mon.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (MON\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List of Overtime Phantom Blocks* for i in range(0, len(instructors)):  
 if str(instructors[i].get\_number()) == str(INSTRUCTOR\_ID.get().split(sep, 1)[0]):  
 for j in range(0, len(timepoints)):  
 if (str(MON\_END.get()) == str(MonStart[i][0])):  
 if (str(MON\_END.get()) == str(MonStart[i][0])):  
 if (MON\_END.get() == timepoints[j].get\_point()):  
 over\_phantom\_time.append(timepoints[j].get\_phantomBlocks())  
 elif (str(MON\_END.get()) != str(MonStart[i][0])):  
 if len(Time\_Mon) != 0:  
 if (MON\_START.get() == timepoints[j].get\_point()):  
 over\_phantom\_time.append(timepoints[j].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Mon)):  
 if (Time\_Mon[i] in startingTime\_Mon[0]):  
 if (Time\_Mon[i] not in endingTime\_Mon[0]):  
 if (Time\_Mon[i] not in phantom\_time[0]):  
 qualifiedMTs\_Mon.append(ID\_Mon[i])  
 qualifiedBlocks\_Mon.append(Time\_Mon[i])  
 if (Time\_Mon[i] in over\_phantom\_time[0]):  
 qualifiedMTs\_Mon.append(ID\_Mon[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Mon)):  
 if (Time\_Mon[i] in qualifiedBlocks\_Mon):  
 cursor.execute(  
 **"INSERT INTO 'instructor\_overtime' (instructor\_id, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(INSTRUCTOR\_ID.get().split(sep, 1)[0]), str(ID\_Mon[i]), str(MON\_START.get()),  
 str(MON\_END.get())))  
 conn.commit()  
 MON\_START.set(**""**)  
 MON\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 if (not (TUE\_START.get() == **""**)):  
 ID\_Tue=list() *# Meeting Time IDs in Tuesday* Time\_Tue=list()  
 startingTime\_Tue=list()  
 endingTime\_Tue=list()  
 over\_phantom\_time=list()  
 phantom\_time=list()  
 qualifiedMTs\_Tue=list()  
 qualifiedBlocks\_Tue=list()  
 *# List all Tuesdays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Tuesday'** == meetingTimes[i].get\_day()):  
 ID\_Tue.append(meetingTimes[i].get\_id())  
 Time\_Tue.append(meetingTimes[i].get\_time())  
   
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (TUE\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Tue.append(timepoints[i].get\_blocks())  
 if (TUE\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Tue.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (TUE\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List of Overtime Phantom Blocks* for i in range(0, len(instructors)):  
 if str(instructors[i].get\_number()) == str(INSTRUCTOR\_ID.get().split(sep, 1)[0]):  
 for j in range(0, len(timepoints)):  
 if (str(TUE\_END.get()) == str(TueStart[i][0])):  
 if (str(TUE\_END.get()) == str(TueStart[i][0])):  
 if (TUE\_END.get() == timepoints[j].get\_point()):  
 over\_phantom\_time.append(timepoints[j].get\_phantomBlocks())  
 elif (str(TUE\_END.get()) != str(TueStart[i][0])):  
 if len(Time\_Tue) != 0:  
 if (TUE\_START.get() == timepoints[j].get\_point()):  
 over\_phantom\_time.append(timepoints[j].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Tue)):  
 if (Time\_Tue[i] in startingTime\_Tue[0]):  
 if (Time\_Tue[i] not in endingTime\_Tue[0]):  
 if (Time\_Tue[i] not in phantom\_time[0]):  
 qualifiedMTs\_Tue.append(ID\_Tue[i])  
 qualifiedBlocks\_Tue.append(Time\_Tue[i])  
 if (Time\_Tue[i] in over\_phantom\_time[0]):  
 qualifiedMTs\_Tue.append(ID\_Tue[i])  
 qualifiedBlocks\_Tue.append(Time\_Tue[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Tue)):  
 if (Time\_Tue[i] in qualifiedBlocks\_Tue):  
 cursor.execute(  
 **"INSERT INTO 'instructor\_overtime' (instructor\_id, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(INSTRUCTOR\_ID.get().split(sep, 1)[0]), str(ID\_Tue[i]), str(TUE\_START.get()),  
 str(TUE\_END.get())))  
 conn.commit()  
 TUE\_START.set(**""**)  
 TUE\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 if (not (WED\_START.get() == **""**)):  
 ID\_Wed=list() *# Meeting Time IDs in Wednesday* Time\_Wed=list()  
 startingTime\_Wed=list()  
 endingTime\_Wed=list()  
 over\_phantom\_time=list()  
 phantom\_time=list()  
 qualifiedMTs\_Wed=list()  
 qualifiedBlocks\_Wed=list()  
 *# List all Wednesdays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Wednesday'** == meetingTimes[i].get\_day()):  
 ID\_Wed.append(meetingTimes[i].get\_id())  
 Time\_Wed.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (WED\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Wed.append(timepoints[i].get\_blocks())  
 if (WED\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Wed.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (WED\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List of Overtime Phantom Blocks* for i in range(0, len(instructors)):  
 if str(instructors[i].get\_number()) == str(INSTRUCTOR\_ID.get().split(sep, 1)[0]):  
 for j in range(0, len(timepoints)):  
 if (str(WED\_END.get()) == str(WedStart[i][0])):  
 if (str(WED\_END.get()) == str(WedStart[i][0])):  
 if (WED\_END.get() == timepoints[j].get\_point()):  
 over\_phantom\_time.append(timepoints[j].get\_phantomBlocks())  
 elif (str(WED\_END.get()) != str(WedStart[i][0])):  
 if len(Time\_Wed) != 0:  
 if (WED\_START.get() == timepoints[j].get\_point()):  
 over\_phantom\_time.append(timepoints[j].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Wed)):  
 if (Time\_Wed[i] in startingTime\_Wed[0]):  
 if (Time\_Wed[i] not in endingTime\_Wed[0]):  
 if (Time\_Wed[i] not in phantom\_time[0]):  
 qualifiedMTs\_Wed.append(ID\_Wed[i])  
 qualifiedBlocks\_Wed.append(Time\_Wed[i])  
 if (Time\_Wed[i] in over\_phantom\_time[0]):  
 qualifiedMTs\_Wed.append(ID\_Wed[i])  
 qualifiedBlocks\_Wed.append(Time\_Wed[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Wed)):  
 if (Time\_Wed[i] in qualifiedBlocks\_Wed):  
 cursor.execute(  
 **"INSERT INTO 'instructor\_overtime' (instructor\_id, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(INSTRUCTOR\_ID.get().split(sep, 1)[0]), str(ID\_Wed[i]), str(WED\_START.get()),  
 str(WED\_END.get())))  
 conn.commit()  
 WED\_START.set(**""**)  
 WED\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 if (not (THU\_START.get() == **""**)):  
 ID\_Thu=list() *# Meeting Time IDs in Thursday* Time\_Thu=list()  
 startingTime\_Thu=list()  
 endingTime\_Thu=list()  
 over\_phantom\_time=list()  
 phantom\_time=list()  
 qualifiedMTs\_Thu=list()  
 qualifiedBlocks\_Thu=list()  
 *# List all Thursdays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Thursday'** == meetingTimes[i].get\_day()):  
 ID\_Thu.append(meetingTimes[i].get\_id())  
 Time\_Thu.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (THU\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Thu.append(timepoints[i].get\_blocks())  
 if (THU\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Thu.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (THU\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List of Overtime Phantom Blocks* for i in range(0, len(instructors)):  
 if str(instructors[i].get\_number()) == str(INSTRUCTOR\_ID.get().split(sep, 1)[0]):  
 for j in range(0, len(timepoints)):  
 if (str(THU\_END.get()) == str(ThuStart[i][0])):  
 if (str(THU\_END.get()) == str(ThuStart[i][0])):  
 if (THU\_END.get() == timepoints[j].get\_point()):  
 over\_phantom\_time.append(timepoints[j].get\_phantomBlocks())  
 elif (str(THU\_END.get()) != str(ThuStart[i][0])):  
 if len(Time\_Thu) != 0:  
 if (THU\_START.get() == timepoints[j].get\_point()):  
 over\_phantom\_time.append(timepoints[j].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Thu)):  
 if (Time\_Thu[i] in startingTime\_Thu[0]):  
 if (Time\_Thu[i] not in endingTime\_Thu[0]):  
 if (Time\_Thu[i] not in phantom\_time[0]):  
 qualifiedMTs\_Thu.append(ID\_Thu[i])  
 qualifiedBlocks\_Thu.append(Time\_Thu[i])  
 if (Time\_Thu[i] in over\_phantom\_time[0]):  
 qualifiedMTs\_Thu.append(ID\_Thu[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Thu)):  
 if (Time\_Thu[i] in qualifiedBlocks\_Thu):  
 cursor.execute(  
 **"INSERT INTO 'instructor\_overtime' (instructor\_id, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(INSTRUCTOR\_ID.get().split(sep, 1)[0]), str(ID\_Thu[i]), str(THU\_START.get()),  
 str(THU\_END.get())))  
 conn.commit()  
 THU\_START.set(**""**)  
 THU\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 if (not (FRI\_START.get() == **""**)):  
 ID\_Fri=list() *# Meeting Time IDs in Friday* Time\_Fri=list()  
 startingTime\_Fri=list()  
 endingTime\_Fri=list()  
 over\_phantom\_time=list()  
 phantom\_time=list()  
 qualifiedMTs\_Fri=list()  
 qualifiedBlocks\_Fri=list()  
 *# List all Fridays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Friday'** == meetingTimes[i].get\_day()):  
 ID\_Fri.append(meetingTimes[i].get\_id())  
 Time\_Fri.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (FRI\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Fri.append(timepoints[i].get\_blocks())  
 if (FRI\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Fri.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (FRI\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List of Overtime Phantom Blocks* for i in range(0, len(instructors)):  
 if str(instructors[i].get\_number()) == str(INSTRUCTOR\_ID.get().split(sep, 1)[0]):  
 for j in range(0, len(timepoints)):  
 if (str(FRI\_END.get()) == str(FriStart[i][0])):  
 if (str(FRI\_END.get()) == str(FriStart[i][0])):  
 if (FRI\_END.get() == timepoints[j].get\_point()):  
 over\_phantom\_time.append(timepoints[j].get\_phantomBlocks())  
 elif (str(FRI\_END.get()) != str(FriStart[i][0])):  
 if len(Time\_Fri) != 0:  
 if (FRI\_START.get() == timepoints[j].get\_point()):  
 over\_phantom\_time.append(timepoints[j].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Fri)):  
 if (Time\_Fri[i] in startingTime\_Fri[0]):  
 if (Time\_Fri[i] not in endingTime\_Fri[0]):  
 if (Time\_Fri[i] not in phantom\_time[0]):  
 qualifiedMTs\_Fri.append(ID\_Fri[i])  
 qualifiedBlocks\_Fri.append(Time\_Fri[i])  
 if (Time\_Fri[i] in over\_phantom\_time[0]):  
 qualifiedMTs\_Fri.append(ID\_Fri[i])  
 qualifiedBlocks\_Fri.append(Time\_Fri[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Fri)):  
 if (Time\_Fri[i] in qualifiedBlocks\_Fri):  
 cursor.execute(  
 **"INSERT INTO 'instructor\_overtime' (instructor\_id, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(INSTRUCTOR\_ID.get().split(sep, 1)[0]), str(ID\_Fri[i]), str(FRI\_START.get()),  
 str(FRI\_END.get())))  
 conn.commit()  
 FRI\_START.set(**""**)  
 FRI\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 if (not (SAT\_START.get() == **""**)):  
 ID\_Sat=list() *# Meeting Time IDs in Saturday* Time\_Sat=list()  
 startingTime\_Sat=list()  
 endingTime\_Sat=list()  
 over\_phantom\_time=list()  
 phantom\_time=list()  
 qualifiedMTs\_Sat=list()  
 qualifiedBlocks\_Sat=list()  
 *# List all Saturdays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Saturday'** == meetingTimes[i].get\_day()):  
 ID\_Sat.append(meetingTimes[i].get\_id())  
 Time\_Sat.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (SAT\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Sat.append(timepoints[i].get\_blocks())  
 if (SAT\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Sat.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (SAT\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List of Overtime Phantom Blocks* for i in range(0, len(instructors)):  
 if str(instructors[i].get\_number()) == str(INSTRUCTOR\_ID.get().split(sep, 1)[0]):  
 for j in range(0, len(timepoints)):  
 if (str(SAT\_END.get()) == str(SatStart[i][0])):  
 if (str(SAT\_END.get()) == str(SatStart[i][0])):  
 if (SAT\_END.get() == timepoints[j].get\_point()):  
 over\_phantom\_time.append(timepoints[j].get\_phantomBlocks())  
 elif (str(SAT\_END.get()) != str(SatStart[i][0])):  
 if len(Time\_Sat) != 0:  
 if (SAT\_START.get() == timepoints[j].get\_point()):  
 over\_phantom\_time.append(timepoints[j].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Sat)):  
 if (Time\_Sat[i] in startingTime\_Sat[0]):  
 if (Time\_Sat[i] not in endingTime\_Sat[0]):  
 if (Time\_Sat[i] not in phantom\_time[0]):  
 qualifiedMTs\_Sat.append(ID\_Sat[i])  
 qualifiedBlocks\_Sat.append(Time\_Sat[i])  
 if (Time\_Sat[i] in over\_phantom\_time[0]):  
 qualifiedMTs\_Sat.append(ID\_Sat[i])  
 qualifiedBlocks\_Sat.append(Time\_Sat[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Sat)):  
 if (Time\_Sat[i] in qualifiedBlocks\_Sat):  
 cursor.execute(  
 **"INSERT INTO 'instructor\_overtime' (instructor\_id, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(INSTRUCTOR\_ID.get().split(sep, 1)[0]), str(ID\_Sat[i]), str(SAT\_START.get()),  
 str(SAT\_END.get())))  
 conn.commit()  
 SAT\_START.set(**""**)  
 SAT\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 if (not (SUN\_START.get() == **""**)):  
 ID\_Sun=list() *# Meeting Time IDs in Sunday* Time\_Sun=list()  
 startingTime\_Sun=list()  
 endingTime\_Sun=list()  
 over\_phantom\_time=list()  
 phantom\_time=list()  
 qualifiedMTs\_Sun=list()  
 qualifiedBlocks\_Sun=list()  
 *# List all Sundays' MTs* for i in range(0, len(meetingTimes)):  
 if (**'Sunday'** == meetingTimes[i].get\_day()):  
 ID\_Sun.append(meetingTimes[i].get\_id())  
 Time\_Sun.append(meetingTimes[i].get\_time())  
 *# List Start Blocks and End Blocks* for i in range(0, len(timepoints)):  
 if (SUN\_START.get() == timepoints[i].get\_point()):  
 startingTime\_Sun.append(timepoints[i].get\_blocks())  
 if (SUN\_END.get() == timepoints[i].get\_point()):  
 endingTime\_Sun.append(timepoints[i].get\_blocks())  
 *# List of Phantom Blocks* for i in range(0, len(timepoints)):  
 if (SUN\_END.get() == timepoints[i].get\_point()):  
 phantom\_time.append(timepoints[i].get\_phantomBlocks())  
 *# List of Overtime Phantom Blocks* for i in range(0, len(instructors)):  
 if str(instructors[i].get\_number()) == str(INSTRUCTOR\_ID.get().split(sep, 1)[0]):  
 for j in range(0, len(timepoints)):  
 if (str(SUN\_END.get()) == str(SunStart[i][0])):  
 if (str(SUN\_END.get()) == str(SunStart[i][0])):  
 if (SUN\_END.get() == timepoints[j].get\_point()):  
 over\_phantom\_time.append(timepoints[j].get\_phantomBlocks())  
 elif (str(SUN\_END.get()) != str(SunStart[i][0])):  
 if len(Time\_Sun) != 0:  
 if (SUN\_START.get() == timepoints[j].get\_point()):  
 over\_phantom\_time.append(timepoints[j].get\_phantomBlocks())  
 *# List Qualified Blocks* for i in range(0, len(Time\_Sun)):  
 if (Time\_Sun[i] in startingTime\_Sun[0]):  
 if (Time\_Sun[i] not in endingTime\_Sun[0]):  
 if (Time\_Sun[i] not in phantom\_time[0]):  
 qualifiedMTs\_Sun.append(ID\_Sun[i])  
 qualifiedBlocks\_Sun.append(Time\_Sun[i])  
 if (Time\_Sun[i] in over\_phantom\_time[0]):  
 qualifiedMTs\_Sun.append(ID\_Sun[i])  
 qualifiedBlocks\_Sun.append(Time\_Sun[i])  
 *# Add Qualified Blocks to DB* ARADB()  
 for i in range(0, len(Time\_Sun)):  
 if (Time\_Sun[i] in qualifiedBlocks\_Sun):  
 cursor.execute(  
 **"INSERT INTO 'instructor\_overtime' (instructor\_id, meeting\_time\_id, start, end) VALUES(?, ?, ?, ?)"**,  
 (str(INSTRUCTOR\_ID.get().split(sep, 1)[0]), str(ID\_Sun[i]), str(SUN\_START.get()),  
 str(SUN\_END.get())))  
 conn.commit()  
 SUN\_START.set(**""**)  
 SUN\_END.set(**""**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Created a data!"**, fg = **"green"**)  
  
 def View():  
 tree.delete(\*tree.get\_children())  
 ARADB()  
 cursor.execute(**"SELECT \* FROM `instructor\_overtime` ORDER BY `instructor\_id` ASC"**)  
 fetch=cursor.fetchall()  
 for data in fetch:  
 tree.insert(**''**, **'end'**, values = (data[0], data[1], data[2], data[3]))  
 cursor.close()  
 conn.close()  
 txt\_result.config(text = **"Successfully viewed the data from database"**, fg = **"black"**)  
  
 def Delete():  
 if INSTRUCTOR\_ID.get() == **""**:  
 txt\_result.config(text = **"Please choose an instructor!"**, fg = **"red"**)  
 else:  
 sep=**' '** ARADB()  
 cursor.execute(  
 **"DELETE FROM 'instructor\_overtime' WHERE instructor\_id = '"** + str(  
 INSTRUCTOR\_ID.get().split(sep, 1)[0]) + **"'"**)  
 conn.commit()  
 cb\_instructor\_id.delete(0, **'end'**)  
 cb\_meeting\_time.delete(0, **'end'**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Deleted Successfully!"**, fg = **"green"**)  
  
 def Clear():  
 result=messagebox.askquestion(**"Clear"**, **"This will clear the Instructor's Overtime table.**\n**Do you want to proceed?"**, icon = **'warning'**)  
 if result == **'yes'**:  
 ARADB()  
 cursor.execute(**"""DROP TABLE instructor\_overtime"""**)  
 cursor.execute(**"""create table instructor\_overtime (instructor\_id text NOT NULL, meeting\_time\_id text NOT NULL, start text, end text)"""**)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Table Cleared Successfully!"**, fg = **"green"**)  
 else:  
 pass  
  
 def Refresh():  
 IO.destroy()  
 Modify\_Overtime()  
  
 *# ==================================VARIABLES==========================================* INSTRUCTOR\_ID=StringVar(IO)  
 MEETING\_TIME=StringVar(IO)  
 MON\_START=StringVar(IO)  
 MON\_END=StringVar(IO)  
 TUE\_START=StringVar(IO)  
 TUE\_END=StringVar(IO)  
 WED\_START=StringVar(IO)  
 WED\_END=StringVar(IO)  
 THU\_START=StringVar(IO)  
 THU\_END=StringVar(IO)  
 FRI\_START=StringVar(IO)  
 FRI\_END=StringVar(IO)  
 SAT\_START=StringVar(IO)  
 SAT\_END=StringVar(IO)  
 SUN\_START=StringVar(IO)  
 SUN\_END=StringVar(IO)  
  
 *# ==================================FRAME==============================================* Top=Frame(IO, width = 900, height = 50, bd = 8, relief = **"raise"**, bg = **"#9EE09E"**)  
 Top.pack(side = TOP)  
 Left=Frame(IO, width = 300, height = 500, bd = 8, relief = **"raise"**)  
 Left.pack(side = LEFT)  
 Right=Frame(IO, width = 600, height = 500, bd = 8, relief = **"raise"**)  
 Right.pack(side = RIGHT)  
 Forms=Frame(Left, width = 300, height = 450)  
 Forms.pack(side = TOP)  
 Buttons=Frame(Left, width = 300, height = 100, bd = 8, relief = **"raise"**)  
 Buttons.pack(side = BOTTOM)  
  
 *# ==================================LABEL WIDGET=======================================* txt\_title=Label(Top, width = 900, font = (**'arial'**, 24), text = **"Instructor Overtime"**, fg = **"Black"**,  
 bg = **"#B3EE9A"**)   
 txt\_title.pack()  
 txt\_instructor=Label(Forms, text = **"Select Instructor:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_instructor.grid(row = 0, stick = **"e"**)  
  
 txt\_monday=Label(Forms, text = **"Monday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_monday.grid(row = 1, stick = **"e"**)  
 txt\_tuesday=Label(Forms, text = **"Tuesday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_tuesday.grid(row = 2, stick = **"e"**)  
 txt\_wednesday=Label(Forms, text = **"Wednesday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_wednesday.grid(row = 3, stick = **"e"**)  
 txt\_thursday=Label(Forms, text = **"Thursday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_thursday.grid(row = 4, stick = **"e"**)  
 txt\_friday=Label(Forms, text = **"Friday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_friday.grid(row = 5, stick = **"e"**)  
 txt\_saturday=Label(Forms, text = **"Saturday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_saturday.grid(row = 6, stick = **"e"**)  
 txt\_sunday=Label(Forms, text = **"Sunday"**, font = (**'arial'**, 16, **'bold'**), bd = 15)  
 txt\_sunday.grid(row = 7, stick = **"e"**)  
 txt\_result=Label(Buttons)  
 txt\_result.pack(side = TOP)  
  
 *# ==================================ENTRY WIDGET=======================================* cb\_instructor\_id=ttk.Combobox(Forms, textvariable = INSTRUCTOR\_ID, width = 44)  
 cb\_instructor\_id[**'values'**]=combobox\_input0()  
 cb\_instructor\_id.place(x = 210, y = 18)  
 cb\_meeting\_time=ttk.Combobox(Forms, textvariable = MEETING\_TIME, width = 20)  
 cb\_meeting\_time[**'values'**]=combobox\_input1()  
  
 txt\_monstart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_monstart.grid(column = 1, row = 1, stick = **"w"**)  
 cb\_monstart=ttk.Combobox(Forms, textvariable = MON\_START, width = 10)  
 cb\_monstart[**'values'**]=combobox\_input2()  
 cb\_monstart.grid(column = 2, row = 1, stick = **"w"**)  
 txt\_monend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 12)  
 txt\_monend.grid(column = 3, row = 1, stick = **"w"**)  
 cb\_monend=ttk.Combobox(Forms, textvariable = MON\_END, width = 10)  
 cb\_monend[**'values'**]=combobox\_input2()  
 cb\_monend.grid(column = 4, row = 1, padx = 10)  
  
 txt\_tuestart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_tuestart.grid(column = 1, row = 2, stick = **"w"**)  
 cb\_tuestart=ttk.Combobox(Forms, textvariable = TUE\_START, width = 10)  
 cb\_tuestart[**'values'**]=combobox\_input2()  
 cb\_tuestart.grid(column = 2, row = 2, stick = **"w"**)  
 txt\_tueend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_tueend.grid(column = 3, row = 2, stick = **"w"**)  
 cb\_tueend=ttk.Combobox(Forms, textvariable = TUE\_END, width = 10)  
 cb\_tueend[**'values'**]=combobox\_input2()  
 cb\_tueend.grid(column = 4, row = 2, padx = 10)  
  
 txt\_wedstart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_wedstart.grid(column = 1, row = 3, stick = **"w"**)  
 cb\_wedstart=ttk.Combobox(Forms, textvariable = WED\_START, width = 10)  
 cb\_wedstart[**'values'**]=combobox\_input2()  
 cb\_wedstart.grid(column = 2, row = 3, stick = **"w"**)  
 txt\_wedend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_wedend.grid(column = 3, row = 3, stick = **"w"**)  
 cb\_wedend=ttk.Combobox(Forms, textvariable = WED\_END, width = 10)  
 cb\_wedend[**'values'**]=combobox\_input2()  
 cb\_wedend.grid(column = 4, row = 3, padx = 10)  
  
 txt\_thustart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_thustart.grid(column = 1, row = 4, stick = **"w"**)  
 cb\_thustart=ttk.Combobox(Forms, textvariable = THU\_START, width = 10)  
 cb\_thustart[**'values'**]=combobox\_input2()  
 cb\_thustart.grid(column = 2, row = 4, stick = **"w"**)  
 txt\_thuend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_thuend.grid(column = 3, row = 4, stick = **"w"**)  
 cb\_thuend=ttk.Combobox(Forms, textvariable = THU\_END, width = 10)  
 cb\_thuend[**'values'**]=combobox\_input2()  
 cb\_thuend.grid(column = 4, row = 4, padx = 10)  
  
 txt\_fristart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_fristart.grid(column = 1, row = 5, stick = **"w"**)  
 cb\_fristart=ttk.Combobox(Forms, textvariable = FRI\_START, width = 10)  
 cb\_fristart[**'values'**]=combobox\_input2()  
 cb\_fristart.grid(column = 2, row = 5)  
 txt\_friend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_friend.grid(column = 3, row = 5, stick = **"w"**)  
 cb\_friend=ttk.Combobox(Forms, textvariable = FRI\_END, width = 10)  
 cb\_friend[**'values'**]=combobox\_input2()  
 cb\_friend.grid(column = 4, row = 5, padx = 10)  
  
 txt\_satstart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_satstart.grid(column = 1, row = 6, stick = **"w"**)  
 cb\_satstart=ttk.Combobox(Forms, textvariable = SAT\_START, width = 10)  
 cb\_satstart[**'values'**]=combobox\_input2()  
 cb\_satstart.grid(column = 2, row = 6, stick = **"w"**)  
 txt\_satend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_satend.grid(column = 3, row = 6, stick = **"w"**)  
 cb\_satend=ttk.Combobox(Forms, textvariable = SAT\_END, width = 10)  
 cb\_satend[**'values'**]=combobox\_input2()  
 cb\_satend.grid(column = 4, row = 6, padx = 10)  
  
 txt\_sunstart=Label(Forms, text = **"Start:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_sunstart.grid(column = 1, row = 7, stick = **"w"**)  
 cb\_sunstart=ttk.Combobox(Forms, textvariable = SUN\_START, width = 10)  
 cb\_sunstart[**'values'**]=combobox\_input2()  
 cb\_sunstart.grid(column = 2, row = 7, stick = **"w"**)  
 txt\_sunend=Label(Forms, text = **"End:"**, font = (**'arial'**, 12), bd = 15)  
 txt\_sunend.grid(column = 3, row = 7, stick = **"w"**)  
 cb\_sunend=ttk.Combobox(Forms, textvariable = SUN\_END, width = 10)  
 cb\_sunend[**'values'**]=combobox\_input2()  
 cb\_sunend.grid(column = 4, row = 7, padx = 10)  
  
 *# ==================================BUTTONS WIDGET=====================================* btn\_create=Button(Buttons, width = 10, text = **"Create"**, command = Create)  
 btn\_create.pack(side = LEFT)  
 btn\_view=Button(Buttons, width = 10, text = **"View"**, command = View)  
 btn\_view.pack(side = LEFT)  
 btn\_update=Button(Buttons, width = 10, text = **"Update"**, state = DISABLED)  
 btn\_update.pack(side = LEFT)  
 btn\_delete=Button(Buttons, width = 10, text = **"Delete"**, command = Delete)  
 btn\_delete.pack(side = LEFT)  
 btn\_clear=Button(Buttons, width = 10, text = **"Clear"**, fg = **"Black"**, bg = **"#F2B6AE"**, command = Clear)  
 btn\_clear.pack(side = LEFT)  
 btn\_refresh=Button(Top, width = 10, text = **"Refresh (F5)"**, font = (**'arial'**, 10, **'bold'**), fg = **"White"**,  
 bg = **"green"**, command = Refresh)  
 btn\_refresh.pack(side = RIGHT)  
  
 *# ==================================LIST WIDGET========================================* scrollbary=Scrollbar(Right, orient = VERTICAL)  
 scrollbarx=Scrollbar(Right, orient = HORIZONTAL)  
 columns=(**"Instructor"**, **"Meeting Time"**, **"Start"**, **"End"**)  
 tree=ttk.Treeview(Right, columns = columns, selectmode = **"extended"**, height = 500,  
 yscrollcommand = scrollbary.set, xscrollcommand = scrollbarx.set)  
  
 def treeview\_sort\_column(tree, col, reverse):  
 l=[(tree.set(k, col), k) for k in tree.get\_children(**''**)]  
 l.sort(reverse = reverse)  
  
 *# rearrange items in sorted positions* for index, (val, k) in enumerate(l):  
 tree.move(k, **''**, index)  
  
 *# reverse sort next time* tree.heading(col, command = lambda \_col=col: treeview\_sort\_column(tree, \_col, not reverse))  
  
 for col in columns:  
 tree.heading(col, text = col, command = lambda \_col=col: \  
 treeview\_sort\_column(tree, \_col, False))  
 *# for TREE Scroll Bars* scrollbary.config(command = tree.yview)  
 scrollbary.pack(side = RIGHT, fill = Y)  
 scrollbarx.config(command = tree.xview)  
 scrollbarx.pack(side = BOTTOM, fill = X)  
 *# inside the tree view* tree.heading(**'Instructor'**, text = **"Instructor"**, anchor = W)  
 tree.heading(**'Meeting Time'**, text = **"Meeting Time"**, anchor = W)  
 tree.heading(**'Start'**, text = **"Start"**, anchor = W)  
 tree.heading(**'End'**, text = **"End"**, anchor = W)  
 tree.column(**'#0'**, stretch = NO, minwidth = 0, width = 0)  
 tree.column(**'#1'**, stretch = NO, minwidth = 75, width = 75)  
 tree.column(**'#2'**, stretch = NO, minwidth = 150, width = 150)  
 tree.column(**'#3'**, stretch = NO, minwidth = 60, width = 60)  
 tree.bind(**'<ButtonRelease-1>'**, click)  
 tree.pack()  
  
 if \_\_name\_\_ == **'\_\_main\_\_'**:  
 View()  
 IO.mainloop()  
  
 def Assign\_Subject():  
 AS=Toplevel(instroot)  
 AS.title(**"ECE-SPV"**)  
 AS.wm\_iconbitmap(**"ece-spv.ico"**)  
 AS.screen\_width=AS.winfo\_screenwidth()  
 AS.screen\_height=AS.winfo\_screenheight()  
 AS.width=720  
 AS.height=627  
 AS.x=(screen\_width / 2) - (width / 2)  
 AS.y=(screen\_height / 2) - (height / 2)  
 AS.geometry(**'%dx%d+%d+%d'** % (width, height, x, y))  
 AS.resizable(0, 0)  
  
 *# ==================================COMMANDS============================================* def click(event):  
 try:  
 selected=tree.focus() *# Grab record position/number* values=tree.item(selected, **'values'**) *# Grab record values* cb\_subject\_code.delete(0, **'end'**)  
 cb\_instructor\_number.delete(0, **'end'**)  
 cb\_subject\_code.insert(0, values[0])  
 cb\_instructor\_number.insert(0, values[1])  
 except IndexError: pass  
  
 def combobox\_input0():  
 ARADB()  
 cursor.execute(**"SELECT \* FROM `subject`"**)  
 conn.commit()  
 data=[]  
 for row in cursor.fetchall():  
 data.append(str(row[0]) + **' ['** + str(row[1]) + **']'**)  
 return data  
 cursor.close()  
 conn.close()  
  
 def combobox\_input1():  
 ARADB()  
 cursor.execute(**"SELECT \* FROM `instructor`"**)  
 conn.commit()  
 data=[]  
 for row in cursor.fetchall():  
 data.append(str(row[0]) + **' ['** + str(row[1]) + **']'**)  
 return data  
 cursor.close()  
 conn.close()  
  
 def Create():  
 if SUBJECT\_CODE.get() == **""** or INSTRUCTOR\_NUMBER.get() == **""**:  
 txt\_result.config(text = **"Please complete all fields!"**, fg = **"red"**)  
 else:  
 sep=**' '** ARADB()  
 cursor.execute(**"INSERT INTO `subject\_instructor` (subject\_code, instructor\_number) VALUES(?, ?)"**,  
 (  
 str(SUBJECT\_CODE.get().split(sep, 1)[0]), str(INSTRUCTOR\_NUMBER.get().split(sep, 1)[0])))  
 conn.commit()  
 SUBJECT\_CODE.set(**""**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Created a data!"**, fg = **"green"**)  
  
 def View():  
 tree.delete(\*tree.get\_children())  
 ARADB()  
 cursor.execute(**"SELECT \* FROM `subject\_instructor` ORDER BY `instructor\_number` ASC"**)  
 fetch=cursor.fetchall()  
 for data in fetch:  
 tree.insert(**''**, **'end'**, values = (data[0], data[1]))  
 cursor.close()  
 conn.close()  
 txt\_result.config(text = **"Successfully viewed the data from database"**, fg = **"black"**)  
  
 def Delete():  
 if SUBJECT\_CODE.get() == **""** or INSTRUCTOR\_NUMBER.get() == **""**:  
 txt\_result.config(text = **"Please complete all fields!"**, fg = **"red"**)  
 else:  
 sep=**' '** ARADB()  
 cursor.execute(  
 **"DELETE FROM 'subject\_instructor' WHERE subject\_code = '"** + str(  
 SUBJECT\_CODE.get().split(sep, 1)[0]) + **"' and instructor\_number = '"** + str(  
 INSTRUCTOR\_NUMBER.get().split(sep, 1)[0]) + **"'"**)  
 conn.commit()  
 cb\_subject\_code.delete(0, **'end'**)  
 cb\_instructor\_number.delete(0, **'end'**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Deleted Successfully!"**, fg = **"green"**)  
  
 def Clear():  
 result=messagebox.askquestion(**"Clear"**, **"This will clear the Instructor's Subject table.**\n**Do you want to proceed?"**, icon = **'warning'**)  
 if result == **'yes'**:  
 ARADB()  
 cursor.execute(**"""DROP TABLE subject\_instructor"""**)  
 cursor.execute(**"""create table subject\_instructor (subject\_code text NOT NULL, instructor\_number text NOT NULL)"""**)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Table Cleared Successfully!"**, fg = **"green"**)  
 else:  
 pass  
  
 def Refresh():  
 AS.destroy()  
 Assign\_Subject()  
  
 *# ==================================VARIABLES==========================================* SUBJECT\_CODE=StringVar(AS)  
 INSTRUCTOR\_NUMBER=StringVar(AS)  
  
 *# ==================================FRAME==============================================* Top=Frame(AS, width = 900, height = 50, bd = 8, relief = **"raise"**, bg = **"#9EE09E"**)  
 Top.pack(side = TOP)  
 Left=Frame(AS, width = 300, height = 500, bd = 8, relief = **"raise"**)  
 Left.pack(side = LEFT)  
 Right=Frame(AS, width = 600, height = 500, bd = 8, relief = **"raise"**)  
 Right.pack(side = RIGHT)  
 Forms=Frame(Left, width = 300, height = 450)  
 Forms.pack(side = TOP)  
 Buttons=Frame(Left, width = 300, height = 100, bd = 8, relief = **"raise"**)  
 Buttons.pack(side = BOTTOM)  
  
 *# ==================================LABEL WIDGET=======================================* txt\_title=Label(Top, width = 900, font = (**'arial'**, 24), text = **"Instructor's Subject"**, fg = **"Black"**,  
 bg = **"#9EE0BF"**)   
 txt\_title.pack()  
 txt\_subject\_code=Label(Forms, text = **"Select Subject:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_subject\_code.grid(row = 0, stick = **"e"**)  
 txt\_instructor\_number=Label(Forms, text = **"Select Instructor:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_instructor\_number.grid(row = 1, stick = **"e"**)  
 txt\_result=Label(Buttons)  
 txt\_result.pack(side = TOP)  
  
 *# ==================================ENTRY WIDGET=======================================* cb\_subject\_code=ttk.Combobox(Forms, textvariable = SUBJECT\_CODE, width = 50)  
 cb\_subject\_code[**'values'**]=combobox\_input0()  
 cb\_subject\_code.grid(column = 1, row = 0)  
 cb\_instructor\_number=ttk.Combobox(Forms, textvariable = INSTRUCTOR\_NUMBER, width = 50)  
 cb\_instructor\_number[**'values'**]=combobox\_input1()  
 cb\_instructor\_number.grid(column = 1, row = 1)  
  
 *# ==================================BUTTONS WIDGET=====================================* btn\_create=Button(Buttons, width = 10, text = **"Create"**, command = Create)  
 btn\_create.pack(side = LEFT)  
 btn\_view=Button(Buttons, width = 10, text = **"View"**, command = View)  
 btn\_view.pack(side = LEFT)  
 btn\_update=Button(Buttons, width = 10, text = **"Update"**, state = DISABLED)  
 btn\_update.pack(side = LEFT)  
 btn\_delete=Button(Buttons, width = 10, text = **"Delete"**, command = Delete)  
 btn\_delete.pack(side = LEFT)  
 btn\_clear=Button(Buttons, width = 10, text = **"Clear"**, fg = **"Black"**, bg = **"#F2B6AE"**, command = Clear)  
 btn\_clear.pack(side = LEFT)  
 btn\_refresh=Button(Top, width = 10, text = **"Refresh (F5)"**, font = (**'arial'**, 10, **'bold'**), fg = **"White"**,  
 bg = **"green"**, command = Refresh)  
 btn\_refresh.pack(side = RIGHT)  
  
 *# ==================================LIST WIDGET========================================* scrollbary=Scrollbar(Right, orient = VERTICAL)  
 scrollbarx=Scrollbar(Right, orient = HORIZONTAL)  
 columns=(**"Subject Code"**, **"Instructor Number"**)  
 tree=ttk.Treeview(Right, columns = columns, selectmode = **"extended"**, height = 500,  
 yscrollcommand = scrollbary.set, xscrollcommand = scrollbarx.set)  
  
 def treeview\_sort\_column(tree, col, reverse):  
 l=[(tree.set(k, col), k) for k in tree.get\_children(**''**)]  
 l.sort(reverse = reverse)  
  
 *# rearrange items in sorted positions* for index, (val, k) in enumerate(l):  
 tree.move(k, **''**, index)  
  
 *# reverse sort next time* tree.heading(col, command = lambda \_col=col: treeview\_sort\_column(tree, \_col, not reverse))  
  
 for col in columns:  
 tree.heading(col, text = col, command = lambda \_col=col: \  
 treeview\_sort\_column(tree, \_col, False))  
 *# for TREE Scroll Bars* scrollbary.config(command = tree.yview)  
 scrollbary.pack(side = RIGHT, fill = Y)  
 scrollbarx.config(command = tree.xview)  
 scrollbarx.pack(side = BOTTOM, fill = X)  
 *# inside the tree view* tree.heading(**'Subject Code'**, text = **"Subject Code"**, anchor = W)  
 tree.heading(**'Instructor Number'**, text = **"Instructor Number"**, anchor = W)  
 tree.column(**'#0'**, stretch = NO, minwidth = 0, width = 0)  
 tree.column(**'#1'**, stretch = NO, minwidth = 150, width = 150)  
 tree.bind(**'<ButtonRelease-1>'**, click)  
 tree.pack()  
  
 if \_\_name\_\_ == **'\_\_main\_\_'**:  
 View()  
 AS.mainloop()  
  
 def Refresh():  
 instroot.destroy()  
 Instructor()  
  
 *# ==================================VARIABLES==========================================* NUMBER=StringVar(instroot)  
 NAME=StringVar(instroot)  
 MAX\_HOURS=IntVar(instroot)  
 SENIORITY=IntVar(instroot)  
  
 *# ==================================FRAME==============================================* Top=Frame(instroot, width = 900, height = 50, bd = 8, relief = **"raise"**, bg = **"#9EE09E"**)  
 Top.pack(side = TOP)  
 Left=Frame(instroot, width = 300, height = 500, bd = 8, relief = **"raise"**)  
 Left.pack(side = LEFT)  
 Right=Frame(instroot, width = 600, height = 500, bd = 8, relief = **"raise"**)  
 Right.pack(side = RIGHT)  
 Forms=Frame(Left, width = 300, height = 450)  
 Forms.pack(side = TOP)  
 Buttons=Frame(Left, width = 300, height = 100, bd = 8, relief = **"raise"**)  
 Buttons.pack(side = BOTTOM)  
  
 *# ==================================LABEL WIDGET=======================================* txt\_title=Label(Top, width = 900, font = (**'arial'**, 24), text = **"Instructor"**, fg = **"Black"**,  
 bg = **"#B4F6A4"**)   
 txt\_title.pack()  
 txt\_number=Label(Forms, text = **"ID Number:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_number.grid(row = 0, stick = **"e"**)  
 txt\_name=Label(Forms, text = **"Name:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_name.grid(row = 1, stick = **"e"**)  
 txt\_max\_hours=Label(Forms, text = **"Max Units:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_max\_hours.grid(row = 2, stick = **"e"**)  
 txt\_seniority=Label(Forms, text = **"Seniority:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_seniority.grid(row = 3, stick = **"e"**)  
 txt\_result=Label(Buttons)  
 txt\_result.pack(side = TOP)  
  
 *# ==================================ENTRY WIDGET=======================================* cb\_number=ttk.Combobox(Forms, textvariable = NUMBER, width = 20)  
 cb\_number[**'values'**]=combobox\_input()  
 cb\_number.grid(column = 1, row = 0)  
 e\_name=Entry(Forms, textvariable = NAME, width = 20)  
 e\_name.grid(row = 1, column = 1)  
 e\_max\_hours=Entry(Forms, textvariable = MAX\_HOURS, width = 20)  
 e\_max\_hours.grid(row = 2, column = 1)  
 e\_seniority=Entry(Forms, textvariable = SENIORITY, width = 20)  
 e\_seniority.grid(row = 3, column = 1)  
 *# RadioGroup.grid(row=2, column=1)  
  
 # ==================================BUTTONS WIDGET=====================================* btn\_create=Button(Buttons, width = 10, text = **"Create"**, command = Create)  
 btn\_create.pack(side = LEFT)  
 btn\_view=Button(Buttons, width = 10, text = **"View"**, command = View)  
 btn\_view.pack(side = LEFT)  
 btn\_update=Button(Buttons, width = 10, text = **"Update"**, command = Update)  
 btn\_update.pack(side = LEFT)  
 btn\_delete=Button(Buttons, width = 10, text = **"Delete"**, command = Delete)  
 btn\_delete.pack(side = LEFT)  
 btn\_clear=Button(Buttons, width = 10, text = **"Clear"**, fg = **"Black"**, bg = **"#F2B6AE"**, command = Clear)  
 btn\_clear.pack(side = LEFT)  
 *# Upper Button* btn\_modify\_availability=Button(Top, width = 25, text = **"Modify Availability"**, font = (**'arial'**, 10, **'bold'**),  
 fg = **"Black"**, bg = **"#BFE09E"**,  
 command = Modify\_Availability)   
 btn\_modify\_availability.pack(side = LEFT)  
 btn\_modify\_overtime=Button(Top, width = 25, text = **"Modify Overtime"**, font = (**'arial'**, 10, **'bold'**),  
 fg = **"Black"**, bg = **"#B3EE9A"**,  
 command = Modify\_Overtime)   
 btn\_modify\_overtime.pack(side = LEFT)  
 btn\_assign\_subject=Button(Top, width = 25, text = **"Assign Subject"**, font = (**'arial'**, 10, **'bold'**), fg = **"Black"**,  
 bg = **"#9EE0BF"**,  
 command = lambda: Assign\_Subject())   
 btn\_assign\_subject.pack(side = LEFT)  
 btn\_refresh=Button(Top, width = 10, text = **"Refresh (F5)"**, font = (**'arial'**, 10, **'bold'**), fg = **"White"**, bg = **"green"**,  
 command = Refresh)  
 btn\_refresh.pack(side = RIGHT)  
  
 *# ==================================LIST WIDGET========================================* scrollbary=Scrollbar(Right, orient = VERTICAL)  
 scrollbarx=Scrollbar(Right, orient = HORIZONTAL)  
 columns=(**"ID Number"**, **"Name"**, **"Max Units"**, **"Seniority"**)  
 tree=ttk.Treeview(Right, columns = columns, selectmode = **"extended"**, height = 500,  
 yscrollcommand = scrollbary.set, xscrollcommand = scrollbarx.set)  
  
 def treeview\_sort\_column(tree, col, reverse):  
 l=[(tree.set(k, col), k) for k in tree.get\_children(**''**)]  
 l.sort(reverse = reverse)  
  
 *# rearrange items in sorted positions* for index, (val, k) in enumerate(l):  
 tree.move(k, **''**, index)  
  
 *# reverse sort next time* tree.heading(col, command = lambda \_col=col: treeview\_sort\_column(tree, \_col, not reverse))  
  
 for col in columns:  
 tree.heading(col, text = col, command = lambda \_col=col: \  
 treeview\_sort\_column(tree, \_col, False))  
 *# for TREE Scroll Bars* scrollbary.config(command = tree.yview)  
 scrollbary.pack(side = RIGHT, fill = Y)  
 scrollbarx.config(command = tree.xview)  
 scrollbarx.pack(side = BOTTOM, fill = X)  
 *# inside the tree view* tree.heading(**'ID Number'**, text = **"ID Number"**, anchor = W)  
 tree.heading(**'Name'**, text = **"Name"**, anchor = W)  
 tree.heading(**'Max Units'**, text = **"Max Units"**, anchor = W)  
 tree.heading(**'Seniority'**, text = **"Seniority"**, anchor = W)  
 tree.column(**'#0'**, stretch = NO, minwidth = 0, width = 0)  
 tree.column(**'#1'**, stretch = NO, minwidth = 100, width = 100)  
 tree.column(**'#2'**, stretch = NO, minwidth = 200, width = 200)  
 tree.column(**'#3'**, stretch = NO, minwidth = 80, width = 80)  
 tree.bind(**'<ButtonRelease-1>'**, click)  
 tree.pack()  
  
 *# ==================================INITIALIZATION=====================================* if \_\_name\_\_ == **'\_\_main\_\_'**:  
 View()  
 instroot.mainloop()  
  
  
 def Population():  
 POP=Toplevel(root)  
 POP.title(**"ECE-SPV"**)  
 POP.wm\_iconbitmap(**"ece-spv.ico"**)  
 POP.screen\_width=POP.winfo\_screenwidth()  
 POP.screen\_height=POP.winfo\_screenheight()  
 POP.width=720  
 POP.height=627  
 POP.x=(screen\_width / 2) - (width / 2)  
 POP.y=(screen\_height / 2) - (height / 2)  
 POP.geometry(**'%dx%d+%d+%d'** % (width, height, x, y))  
 POP.resizable(0, 0)  
  
 *# ==================================COMMANDS============================================* def click(event):  
 try:  
 selected=tree.focus() *# Grab record position/number* values=tree.item(selected, **'values'**) *# Grab record values* e\_semester.delete(0, **'end'**)  
 e\_number.delete(0, **'end'**)  
 e\_semester.insert(0, values[0])  
 e\_number.insert(0, values[1])  
 except IndexError: pass  
  
 def Create():  
 if SEMESTER.get() == **""** or NUMBER.get() == 0 or NUMBER.get() == **""**:  
 txt\_result.config(text = **"Please complete all fields!"**, fg = **"red"**)  
 else:  
 ARADB()  
 cursor.execute(**"INSERT INTO population (semester, number) VALUES(?, ?)"**,  
 (str(SEMESTER.get()), (str(NUMBER.get())),))  
 conn.commit()  
 SEMESTER.set(**""**)  
 NUMBER.set(**""**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Created a data!"**, fg = **"green"**)  
  
 def View():  
 tree.delete(\*tree.get\_children())  
 ARADB()  
 cursor.execute(**"SELECT \* FROM `population` ORDER BY `semester` DESC"**)  
 fetch=cursor.fetchall()  
 for data in fetch:  
 tree.insert(**''**, **'end'**, values = (data[0], data[1]))  
 cursor.close()  
 conn.close()  
 txt\_result.config(text = **"Successfully viewed the data from database"**, fg = **"black"**)  
  
 def Update():  
 if SEMESTER.get() == **""** or NUMBER.get() == 0 or NUMBER.get() == **""**:  
 txt\_result.config(text=**"Please complete all fields!"**, fg=**"red"**)  
 else:  
 ARADB()  
 cursor.execute(  
 **"UPDATE `population` SET number = :e\_number WHERE semester = :e\_semester"""**, {  
 **'e\_number'**: e\_number.get(), **'e\_semester'**: e\_semester.get()})  
 conn.commit()  
 e\_semester.delete(0, **'end'**)  
 e\_number.delete(0, **'end'**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text=**"Updated Successfully!"**, fg=**"green"**)  
  
 def Delete():  
 if SEMESTER.get() == **""** or NUMBER.get() == 0 or NUMBER.get() == **""**:  
 txt\_result.config(text = **"Please complete all fields!"**, fg = **"red"**)  
 else:  
 ARADB()  
 cursor.execute(**"DELETE FROM 'population' WHERE semester = '"** + str(  
 SEMESTER.get()) + **"' and number = '"** + str(NUMBER.get()) + **"'"**)  
 conn.commit()  
 e\_semester.delete(0, **'end'**)  
 e\_number.delete(0, **'end'**)  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Deleted Successfully!"**, fg = **"green"**)  
  
 def Clear():  
 result=messagebox.askquestion(**"Clear"**, **"This will clear the Population table.**\n**Do you want to proceed?"**, icon = **'warning'**)  
 if result == **'yes'**:  
 ARADB()  
 cursor.execute(**"""DROP TABLE population"""**)  
 cursor.execute(**"""create table population (semester text PRIMARY KEY, number integer NOT NULL)"""**)  
 conn.commit()  
 cursor.close()  
 conn.close()  
 View()  
 txt\_result.config(text = **"Table Cleared Successfully!"**, fg = **"green"**)  
 else:  
 pass  
  
 def Refresh():  
 POP.destroy()  
 Population()  
  
 *# ==================================VARIABLES==========================================* SEMESTER=StringVar(POP)  
 NUMBER=IntVar(POP)  
  
 *# ==================================FRAME==============================================* Top=Frame(POP, width = 900, height = 50, bd = 8, relief = **"raise"**, bg = **"#cc544e"**)  
 Top.pack(side = TOP)  
 Left=Frame(POP, width = 300, height = 500, bd = 8, relief = **"raise"**)  
 Left.pack(side = LEFT)  
 Right=Frame(POP, width = 600, height = 500, bd = 8, relief = **"raise"**)  
 Right.pack(side = RIGHT)  
 Forms=Frame(Left, width = 300, height = 450)  
 Forms.pack(side = TOP)  
 Buttons=Frame(Left, width = 300, height = 100, bd = 8, relief = **"raise"**)  
 Buttons.pack(side = BOTTOM)  
  
 *# ==================================LABEL WIDGET=======================================* txt\_title=Label(Top, width = 900, font = (**'arial'**, 24), text = **"Population"**, fg = **"Black"**,  
 bg = **"#FF6663"**)   
 txt\_title.pack()  
 txt\_semester=Label(Forms, text = **"School-Year, Sem:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_semester.grid(row = 0, stick = **"e"**)  
 txt\_number=Label(Forms, text = **"Number of Students:"**, font = (**'arial'**, 16), bd = 15)  
 txt\_number.grid(row = 1, stick = **"e"**)  
 txt\_result=Label(Buttons)  
 txt\_result.pack(side = TOP)  
  
 *# ==================================ENTRY WIDGET=======================================* e\_semester=Entry(Forms, textvariable = SEMESTER, width = 20)  
 e\_semester.grid(column = 1, row = 0)  
 e\_number=Entry(Forms, textvariable = NUMBER, width = 20)  
 e\_number.grid(column = 1, row = 1)  
  
 *# ==================================BUTTONS WIDGET=====================================* btn\_create=Button(Buttons, width = 10, text = **"Create"**, command = Create)  
 btn\_create.pack(side = LEFT)  
 btn\_view=Button(Buttons, width = 10, text = **"View"**, command = View)  
 btn\_view.pack(side = LEFT)  
 btn\_update=Button(Buttons, width = 10, text = **"Update"**, command = Update)  
 btn\_update.pack(side = LEFT)  
 btn\_delete=Button(Buttons, width = 10, text = **"Delete"**, command = Delete)  
 btn\_delete.pack(side = LEFT)  
 btn\_clear=Button(Buttons, width = 10, text = **"Clear"**, fg = **"Black"**, bg = **"#F2B6AE"**, command = Clear)  
 btn\_clear.pack(side = LEFT)  
 btn\_refresh=Button(Top, width = 10, text = **"Refresh (F5)"**, font = (**'arial'**, 10, **'bold'**), fg = **"White"**, bg = **"green"**,  
 command = Refresh)  
 btn\_refresh.pack(side = RIGHT)  
  
 *# ==================================LIST WIDGET========================================* scrollbary=Scrollbar(Right, orient = VERTICAL)  
 scrollbarx=Scrollbar(Right, orient = HORIZONTAL)  
 columns=(**"Semester"**, **"Number"**)  
 tree=ttk.Treeview(Right, columns = columns, selectmode = **"extended"**, height = 500,  
 yscrollcommand = scrollbary.set, xscrollcommand = scrollbarx.set)  
 *# for TREE Scroll Bars* scrollbary.config(command = tree.yview)  
 scrollbary.pack(side = RIGHT, fill = Y)  
 scrollbarx.config(command = tree.xview)  
 scrollbarx.pack(side = BOTTOM, fill = X)  
 *# inside the tree view* tree.heading(**'Semester'**, text = **"Semester"**, anchor = W)  
 tree.heading(**'Number'**, text = **"Number"**, anchor = W)  
 tree.column(**'#0'**, stretch = NO, minwidth = 0, width = 0)  
 tree.column(**'#1'**, stretch = NO, minwidth = 150, width = 150)  
 tree.bind(**'<ButtonRelease-1>'**, click)  
 tree.pack()  
  
 if \_\_name\_\_ == **'\_\_main\_\_'**:  
 View()  
 POP.mainloop()  
  
  
 try:  
 with open(SETTING\_INFO, **'r'**) as f:  
 if f.read() == **''**:  
 with open(SETTING\_INFO, **'w'**) as f:  
 d\_POPULATION\_SIZE=23  
 d\_MUTATION\_RATE=0.005  
 d\_NUMBER\_OF\_TRIALS=1  
 d\_ALLOWABLE\_ROOM\_CONFLICTS=0.1  
 d\_CONSECUTIVE\_TEACHING\_HOURS=4.5  
 d\_SCHEDULING\_TIME\_LIMIT=12  
 d\_RECURRING\_UPDATE\_TIME=1  
 d\_VERBOSE\_FLAG=False  
 d\_MAX\_ROOM\_LIMIT=False  
 d\_CASUAL\_SPLIT=False  
 d\_UNEQUAL\_SPLIT=False  
 d\_DUAL\_TYPE=True  
 d\_LAB\_VR=False  
 d\_XDEPT\_ROOM\_UTILITY=True  
 d\_INSTRUCTOR\_OVERLOAD=False  
 d\_INSTRUCTOR\_AVAILABILITY=True  
 d\_ROOM\_AVAILABILITY=False  
 d\_SECTION\_AVAILABILITY=False  
 f.write(**"POPULATION\_SIZE="** + str(d\_POPULATION\_SIZE) + **"**\n**"** +  
 **"MUTATION\_RATE="** + str(d\_MUTATION\_RATE) + **"**\n**"** +  
 **"NUMBER\_OF\_TRIALS="** + str(d\_NUMBER\_OF\_TRIALS) + **"**\n**"** +  
 **"ALLOWABLE\_ROOM\_CONFLICTS="** + str(d\_ALLOWABLE\_ROOM\_CONFLICTS) + **"**\n**"** +  
 **"CONSECUTIVE\_TEACHING\_HOURS="** + str(d\_CONSECUTIVE\_TEACHING\_HOURS) + **"**\n**"** +  
 **"SCHEDULING\_TIME\_LIMIT="** + str(d\_SCHEDULING\_TIME\_LIMIT) + **"**\n**"** +  
 **"RECURRING\_UPDATE\_TIME="** + str(d\_RECURRING\_UPDATE\_TIME) + **"**\n**"** +  
 **"VERBOSE\_FLAG="** + str(d\_VERBOSE\_FLAG) + **"**\n**"** +  
 **"ENABLE\_NUMB\_OF\_STUDENTS="** + str(d\_MAX\_ROOM\_LIMIT) + **"**\n**"** +  
 **"DISABLE\_CASUAL\_SPLITTING="** + str(d\_CASUAL\_SPLIT) + **"**\n**"** +  
 **"DISABLE\_UNEQUAL\_SPLIT="** + str(d\_UNEQUAL\_SPLIT) + **"**\n**"** +  
 **"DISABLE\_MIXED\_TYPE="** + str(d\_DUAL\_TYPE) + **"**\n**"** +  
 **"LAB\_ON\_VR="** + str(d\_LAB\_VR) + **"**\n**"** +  
 **"DISABLE\_XDEPT\_ROOM\_UTILITY="** + str(d\_XDEPT\_ROOM\_UTILITY) + **"**\n**"** +  
 **"DISABLE\_INSTRUCTOR\_OVERLOAD="** + str(d\_INSTRUCTOR\_OVERLOAD) + **"**\n**"** +  
 **"LIMIT\_INSTRUCTOR\_AVAILABILITY="** + str(d\_INSTRUCTOR\_AVAILABILITY) + **"**\n**"** +  
 **"LIMIT\_ROOM\_AVAILABILITY="** + str(d\_ROOM\_AVAILABILITY) + **"**\n**"** +  
 **"LIMIT\_SECTION\_AVAILABILITY="** + str(d\_SECTION\_AVAILABILITY))  
 f.close()  
 else: pass  
 f.close()  
 except IOError: *# this is what happens if the file doesn't exist* with open(SETTING\_INFO, **'w'**) as f:  
 d\_POPULATION\_SIZE=23  
 d\_MUTATION\_RATE=0.005  
 d\_NUMBER\_OF\_TRIALS=1  
 d\_ALLOWABLE\_ROOM\_CONFLICTS=0.1  
 d\_CONSECUTIVE\_TEACHING\_HOURS=4.5  
 d\_SCHEDULING\_TIME\_LIMIT=12  
 d\_RECURRING\_UPDATE\_TIME=1  
 d\_VERBOSE\_FLAG=False  
 d\_MAX\_ROOM\_LIMIT=False  
 d\_CASUAL\_SPLIT=False  
 d\_UNEQUAL\_SPLIT=False  
 d\_DUAL\_TYPE=True  
 d\_LAB\_VR=False  
 d\_XDEPT\_ROOM\_UTILITY=True  
 d\_INSTRUCTOR\_OVERLOAD=False  
 d\_INSTRUCTOR\_AVAILABILITY=True  
 d\_ROOM\_AVAILABILITY=False  
 d\_SECTION\_AVAILABILITY=False  
 f.write(**"POPULATION\_SIZE="** + str(d\_POPULATION\_SIZE) + **"**\n**"** +  
 **"MUTATION\_RATE="** + str(d\_MUTATION\_RATE) + **"**\n**"** +  
 **"NUMBER\_OF\_TRIALS="** + str(d\_NUMBER\_OF\_TRIALS) + **"**\n**"** +  
 **"ALLOWABLE\_ROOM\_CONFLICTS="** + str(d\_ALLOWABLE\_ROOM\_CONFLICTS) + **"**\n**"** +  
 **"CONSECUTIVE\_TEACHING\_HOURS="** + str(d\_CONSECUTIVE\_TEACHING\_HOURS) + **"**\n**"** +  
 **"SCHEDULING\_TIME\_LIMIT="** + str(d\_SCHEDULING\_TIME\_LIMIT) + **"**\n**"** +  
 **"RECURRING\_UPDATE\_TIME="** + str(d\_RECURRING\_UPDATE\_TIME) + **"**\n**"** +  
 **"VERBOSE\_FLAG="** + str(d\_VERBOSE\_FLAG) + **"**\n**"** +  
 **"ENABLE\_NUMB\_OF\_STUDENTS="** + str(d\_MAX\_ROOM\_LIMIT) + **"**\n**"** +  
 **"DISABLE\_CASUAL\_SPLITTING="** + str(d\_CASUAL\_SPLIT) + **"**\n**"** +  
 **"DISABLE\_UNEQUAL\_SPLIT="** + str(d\_UNEQUAL\_SPLIT) + **"**\n**"** +  
 **"DISABLE\_MIXED\_TYPE="** + str(d\_DUAL\_TYPE) + **"**\n**"** +  
 **"LAB\_ON\_VR="** + str(d\_LAB\_VR) + **"**\n**"** +  
 **"DISABLE\_XDEPT\_ROOM\_UTILITY="** + str(d\_XDEPT\_ROOM\_UTILITY) + **"**\n**"** +  
 **"DISABLE\_INSTRUCTOR\_OVERLOAD="** + str(d\_INSTRUCTOR\_OVERLOAD) + **"**\n**"** +  
 **"LIMIT\_INSTRUCTOR\_AVAILABILITY="** + str(d\_INSTRUCTOR\_AVAILABILITY) + **"**\n**"** +  
 **"LIMIT\_ROOM\_AVAILABILITY="** + str(d\_ROOM\_AVAILABILITY) + **"**\n**"** +  
 **"LIMIT\_SECTION\_AVAILABILITY="** + str(d\_SECTION\_AVAILABILITY))  
 f.close()  
  
  
 try:  
 with open(OUTPUT\_INFO, **'r'**) as f:  
 if f.read() == **''**:  
 with open(OUTPUT\_INFO, **'w'**) as f:  
 d\_SCHOOL\_NAME=**''** d\_TOP\_HEADER=**''** d\_MID\_HEADER=**''** d\_BOT\_HEADER=**''** d\_UNDERLOGO\_TEXT=**''** d\_COLLEGE=**''** d\_DEPARTMENT=**''** d\_DEPARTMENT\_HEAD=**''** d\_DEAN=**''** d\_INDEX\_NO=**''** d\_ISSUE\_NO=**''** d\_REV\_NO=**''** d\_PAGE\_NO=**''** d\_QAC\_NO=**''** d\_DATE=**''** d\_SCHOOL\_YEAR=**''** d\_SEMESTER=**''** f.write(**'SCHOOL\_NAME='** + str(d\_SCHOOL\_NAME)+**'**\n**'** +  
 **'TOP\_HEADER='** + str(d\_TOP\_HEADER) + **'**\n**'** +  
 **'MID\_HEADER='** + str(d\_MID\_HEADER) + **'**\n**'** +  
 **'BOT\_HEADER='** + str(d\_BOT\_HEADER) + **'**\n**'** +  
 **'UNDERLOGO\_TEXT='** + str(d\_UNDERLOGO\_TEXT) + **'**\n**'** +  
 **'COLLEGE='** + str(d\_COLLEGE) + **'**\n**'** +  
 **'DEPARTMENT='** + str(d\_DEPARTMENT) + **'**\n**'** +  
 **'DEPARTMENT\_HEAD='** + str(d\_DEPARTMENT\_HEAD) + **'**\n**'** +  
 **'DEAN='** + str(d\_DEAN) + **'**\n**'** +  
 **'INDEX\_NO='** + str(d\_INDEX\_NO) + **'**\n**'** +  
 **'ISSUE\_NO='** + str(d\_ISSUE\_NO) + **'**\n**'** +  
 **'REVISION\_NO='** + str(d\_REV\_NO) + **'**\n**'** +  
 **'PAGE\_NO='** + str(d\_PAGE\_NO) + **'**\n**'** +  
 **'QAC\_NO='** + str(d\_QAC\_NO) + **'**\n**'** +  
 **'DATE='** + str(d\_DATE) + **'**\n**'** +  
 **'SCHOOL\_YEAR='** + str(d\_SCHOOL\_YEAR) + **'**\n**'** +  
 **'SEMESTER='** + str(d\_SEMESTER))  
 f.close()  
 else: pass  
 f.close()  
 except IOError: *# this is what happens if the file doesn't exist* with open(OUTPUT\_INFO, **'w'**) as f:  
 d\_SCHOOL\_NAME=**''** d\_TOP\_HEADER=**''** d\_MID\_HEADER=**''** d\_BOT\_HEADER=**''** d\_UNDERLOGO\_TEXT=**''** d\_COLLEGE=**''** d\_DEPARTMENT=**''** d\_DEPARTMENT\_HEAD=**''** d\_DEAN=**''** d\_INDEX\_NO=**''** d\_ISSUE\_NO=**''** d\_REV\_NO=**''** d\_PAGE\_NO=**''** d\_QAC\_NO=**''** d\_DATE=**''** d\_SCHOOL\_YEAR=**''** d\_SEMESTER=**''** f.write(**'SCHOOL\_NAME='** + str(d\_SCHOOL\_NAME)+**'**\n**'** +  
 **'TOP\_HEADER='** + str(d\_TOP\_HEADER) + **'**\n**'** +  
 **'MID\_HEADER='** + str(d\_MID\_HEADER) + **'**\n**'** +  
 **'BOT\_HEADER='** + str(d\_BOT\_HEADER) + **'**\n**'** +  
 **'UNDERLOGO\_TEXT='** + str(d\_UNDERLOGO\_TEXT) + **'**\n**'** +  
 **'COLLEGE='** + str(d\_COLLEGE) + **'**\n**'** +  
 **'DEPARTMENT='** + str(d\_DEPARTMENT) + **'**\n**'** +  
 **'DEPARTMENT\_HEAD='** + str(d\_DEPARTMENT\_HEAD) + **'**\n**'** +  
 **'DEAN='** + str(d\_DEAN) + **'**\n**'** +  
 **'INDEX\_NO='** + str(d\_INDEX\_NO) + **'**\n**'** +  
 **'ISSUE\_NO='** + str(d\_ISSUE\_NO) + **'**\n**'** +  
 **'REVISION\_NO='** + str(d\_REV\_NO) + **'**\n**'** +  
 **'PAGE\_NO='** + str(d\_PAGE\_NO) + **'**\n**'** +  
 **'QAC\_NO='** + str(d\_QAC\_NO) + **'**\n**'** +  
 **'DATE='** + str(d\_DATE) + **'**\n**'** +  
 **'SCHOOL\_YEAR='** + str(d\_SCHOOL\_YEAR) + **'**\n**'** +  
 **'SEMESTER='** + str(d\_SEMESTER))  
 f.close()  
  
  
 def Settings():  
 setroot=Toplevel(root)  
 setroot.title(**"ECE-SPV"**)  
 setroot.wm\_iconbitmap(**"ece-spv.ico"**)  
 screen\_width=setroot.winfo\_screenwidth()  
 screen\_height=setroot.winfo\_screenheight()  
 width=400  
 height=1010  
 x=(screen\_width / 2) - (width / 2)  
 y=(screen\_height / 2) - (height / 2)  
 setroot.geometry(**'%dx%d+%d+%d'** % (width, height, x, y))  
 setroot.resizable(0, 0)  
  
 *# ==================================COMMANDS============================================* def Save():  
 *# # writing new config* with open(SETTING\_INFO, **'w'**) as f:  
 f.write(**"POPULATION\_SIZE="** + str(POPULATION\_SIZE.get()) + **"**\n**"** +  
 **"MUTATION\_RATE="** + str(MUTATION\_RATE.get()) + **"**\n**"** +  
 **"NUMBER\_OF\_TRIALS="** + str(NUMBER\_OF\_TRIALS.get()) + **"**\n**"** +  
 **"ALLOWABLE\_ROOM\_CONFLICTS="** + str(ALLOWABLE\_ROOM\_CONFLICTS.get()) + **"**\n**"** +  
 **"CONSECUTIVE\_TEACHING\_HOURS="** + str(CONSECUTIVE\_TEACHING\_HOURS.get()) + **"**\n**"** +  
 **"SCHEDULING\_TIME\_LIMIT="** + str(SCHEDULING\_TIME\_LIMIT.get()) + **"**\n**"** +  
 **"RECURRING\_UPDATE\_TIME="** + str(RECURRING\_UPDATE\_TIME.get()) + **"**\n**"** +  
 **"VERBOSE\_FLAG="** + str(VERBOSE\_FLAG.get()) + **"**\n**"** +  
 **"ENABLE\_NUMB\_OF\_STUDENTS="** + str(MAX\_ROOM\_LIMIT.get()) + **"**\n**"** +  
 **"DISABLE\_CASUAL\_SPLITTING="** + str(CASUAL\_SPLIT.get()) + **"**\n**"** +  
 **"DISABLE\_UNEQUAL\_SPLIT="** + str(UNEQUAL\_SPLIT.get()) + **"**\n**"** +  
 **"DISABLE\_MIXED\_TYPE="** + str(DUAL\_TYPE.get()) + **"**\n**"** +  
 **"LAB\_ON\_VR="** + str(LAB\_VR.get()) + **"**\n**"** +  
 **"DISABLE\_XDEPT\_ROOM\_UTILITY="** + str(XDEPT\_ROOM\_UTILITY.get()) + **"**\n**"** +  
 **"DISABLE\_INSTRUCTOR\_OVERLOAD="** + str(INSTRUCTOR\_OVERLOAD.get()) + **"**\n**"** +  
 **"LIMIT\_INSTRUCTOR\_AVAILABILITY="** + str(INSTRUCTOR\_AVAILABILITY.get()) + **"**\n**"** +  
 **"LIMIT\_ROOM\_AVAILABILITY="** + str(ROOM\_AVAILABILITY.get()) + **"**\n**"** +  
 **"LIMIT\_SECTION\_AVAILABILITY="** + str(SECTION\_AVAILABILITY.get()))  
 f.close()  
 txt\_result.config(text = **"The changes are saved!"**, fg = **"green"**)  
  
 *# ==================================VARIABLES==========================================* POPULATION\_SIZE=IntVar(setroot)  
 MUTATION\_RATE=DoubleVar(setroot)  
 NUMBER\_OF\_TRIALS=IntVar(setroot)  
 ALLOWABLE\_ROOM\_CONFLICTS=DoubleVar(setroot)  
 CONSECUTIVE\_TEACHING\_HOURS=DoubleVar(setroot)  
 SCHEDULING\_TIME\_LIMIT=DoubleVar(setroot)  
 RECURRING\_UPDATE\_TIME=DoubleVar(setroot)  
 VERBOSE\_FLAG=BooleanVar(setroot)  
 MAX\_ROOM\_LIMIT=BooleanVar(setroot)  
 CASUAL\_SPLIT=BooleanVar(setroot)  
 UNEQUAL\_SPLIT=BooleanVar(setroot)  
 DUAL\_TYPE=BooleanVar(setroot)  
 LAB\_VR=BooleanVar(setroot)  
 XDEPT\_ROOM\_UTILITY=BooleanVar(setroot)  
 INSTRUCTOR\_OVERLOAD=BooleanVar(setroot)  
 INSTRUCTOR\_AVAILABILITY=BooleanVar(setroot)  
 ROOM\_AVAILABILITY=BooleanVar(setroot)  
 SECTION\_AVAILABILITY=BooleanVar(setroot)  
  
 try:  
 with open(SETTING\_INFO, **'r'**) as f:  
 def str2bool(v):  
 return v.lower() in (**"True"**, **"true"**, **"Yes"**, **"yes"**)  
 lines=f.readlines()  
 POPULATION\_SIZE.set(int(lines[0].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 MUTATION\_RATE.set(float(lines[1].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 NUMBER\_OF\_TRIALS.set(int(lines[2].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 ALLOWABLE\_ROOM\_CONFLICTS.set(float(lines[3].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 CONSECUTIVE\_TEACHING\_HOURS.set(float(lines[4].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 SCHEDULING\_TIME\_LIMIT.set(float(lines[5].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 RECURRING\_UPDATE\_TIME.set(float(lines[6].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 VERBOSE\_FLAG.set(str2bool(lines[7].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 MAX\_ROOM\_LIMIT.set(str2bool(lines[8].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 CASUAL\_SPLIT.set(str2bool(lines[9].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 UNEQUAL\_SPLIT.set(str2bool(lines[10].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 DUAL\_TYPE.set(str2bool(lines[11].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 LAB\_VR.set(str2bool(lines[12].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 XDEPT\_ROOM\_UTILITY.set(str2bool(lines[13].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 INSTRUCTOR\_OVERLOAD.set(str2bool(lines[14].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 INSTRUCTOR\_AVAILABILITY.set(str2bool(lines[15].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 ROOM\_AVAILABILITY.set(str2bool(lines[16].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 SECTION\_AVAILABILITY.set(str2bool(lines[17].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 f.close()  
 except IOError: *# this is what happens if the file doesn't exist* with open(SETTING\_INFO, **'w'**) as f:  
 d\_POPULATION\_SIZE=23  
 d\_MUTATION\_RATE=0.005  
 d\_NUMBER\_OF\_TRIALS=1  
 d\_ALLOWABLE\_ROOM\_CONFLICTS=0.1  
 d\_CONSECUTIVE\_TEACHING\_HOURS=4.5  
 d\_SCHEDULING\_TIME\_LIMIT=12  
 d\_RECURRING\_UPDATE\_TIME=1  
 d\_VERBOSE\_FLAG=False  
 d\_MAX\_ROOM\_LIMIT=False  
 d\_CASUAL\_SPLIT=False  
 d\_UNEQUAL\_SPLIT=False  
 d\_DUAL\_TYPE=True  
 d\_LAB\_VR=False  
 d\_XDEPT\_ROOM\_UTILITY=True  
 d\_INSTRUCTOR\_OVERLOAD=False  
 d\_INSTRUCTOR\_AVAILABILITY=True  
 d\_ROOM\_AVAILABILITY=False  
 d\_SECTION\_AVAILABILITY=False  
 f.write(**"POPULATION\_SIZE="** + str(d\_POPULATION\_SIZE) + **"**\n**"** +  
 **"MUTATION\_RATE="** + str(d\_MUTATION\_RATE) + **"**\n**"** +  
 **"NUMBER\_OF\_TRIALS="** + str(d\_NUMBER\_OF\_TRIALS) + **"**\n**"** +  
 **"ALLOWABLE\_ROOM\_CONFLICTS="** + str(d\_ALLOWABLE\_ROOM\_CONFLICTS) + **"**\n**"** +  
 **"CONSECUTIVE\_TEACHING\_HOURS="** + str(d\_CONSECUTIVE\_TEACHING\_HOURS) + **"**\n**"** +  
 **"SCHEDULING\_TIME\_LIMIT="** + str(d\_SCHEDULING\_TIME\_LIMIT) + **"**\n**"** +  
 **"RECURRING\_UPDATE\_TIME="** + str(d\_RECURRING\_UPDATE\_TIME) + **"**\n**"** +  
 **"VERBOSE\_FLAG="** + str(d\_VERBOSE\_FLAG) + **"**\n**"** +  
 **"ENABLE\_NUMB\_OF\_STUDENTS="** + str(d\_MAX\_ROOM\_LIMIT) + **"**\n**"** +  
 **"DISABLE\_CASUAL\_SPLITTING="** + str(d\_CASUAL\_SPLIT) + **"**\n**"** +  
 **"DISABLE\_UNEQUAL\_SPLIT="** + str(d\_UNEQUAL\_SPLIT) + **"**\n**"** +  
 **"DISABLE\_MIXED\_TYPE="** + str(d\_DUAL\_TYPE) + **"**\n**"** +  
 **"LAB\_ON\_VR="** + str(d\_LAB\_VR) + **"**\n**"** +  
 **"DISABLE\_XDEPT\_ROOM\_UTILITY="** + str(d\_XDEPT\_ROOM\_UTILITY) + **"**\n**"** +  
 **"DISABLE\_INSTRUCTOR\_OVERLOAD="** + str(d\_INSTRUCTOR\_OVERLOAD) + **"**\n**"** +  
 **"LIMIT\_INSTRUCTOR\_AVAILABILITY="** + str(d\_INSTRUCTOR\_AVAILABILITY) + **"**\n**"** +  
 **"LIMIT\_ROOM\_AVAILABILITY="** + str(d\_ROOM\_AVAILABILITY) + **"**\n**"** +  
 **"LIMIT\_SECTION\_AVAILABILITY="** + str(d\_SECTION\_AVAILABILITY))  
 f.close()  
  
 *# ==================================FRAME==============================================* Top=Frame(setroot, width = 900, height = 50, bd = 8, relief = **"raise"**, bg = **"#c41e39"**)  
 Top.pack(side = TOP)  
 Left=Frame(setroot, width = 300, height = 500, bd = 8, relief = **"raise"**)  
 Left.pack(side = LEFT)  
 Forms=Frame(Left, width = 300, height = 450)  
 Forms.pack(side = TOP)  
 Buttons=Frame(Left, width = 300, height = 100, bd = 8, relief = **"raise"**)  
 Buttons.pack(side = BOTTOM)  
 rg\_verbose\_flag=Frame(Forms)  
 Yes=Radiobutton(rg\_verbose\_flag, text = **"Yes "**, variable = VERBOSE\_FLAG, value = True, font = (**'arial'**, 10)).pack(side = LEFT)  
 No=Radiobutton(rg\_verbose\_flag, text = **"No "**, variable = VERBOSE\_FLAG, value = False, font = (**'arial'**, 10)).pack(side = LEFT)  
 rg\_max\_room\_limit=Frame(Forms)  
 Yes=Radiobutton(rg\_max\_room\_limit, text = **"Yes "**, variable = MAX\_ROOM\_LIMIT, value = True, font = (**'arial'**, 10)).pack(side = LEFT)  
 No=Radiobutton(rg\_max\_room\_limit, text = **"No "**, variable = MAX\_ROOM\_LIMIT, value = False, font = (**'arial'**, 10)).pack(side = LEFT)  
 rg\_casual\_split=Frame(Forms)  
 Yes=Radiobutton(rg\_casual\_split, text = **"Yes "**, variable = CASUAL\_SPLIT, value = True, font = (**'arial'**, 10)).pack(side = LEFT)  
 No=Radiobutton(rg\_casual\_split, text = **"No "**, variable = CASUAL\_SPLIT, value = False, font = (**'arial'**, 10)).pack(side = LEFT)  
 rg\_unequal\_split=Frame(Forms)  
 Yes=Radiobutton(rg\_unequal\_split, text = **"Yes "**, variable = UNEQUAL\_SPLIT, value = True, font = (**'arial'**, 10)).pack(side = LEFT)  
 No=Radiobutton(rg\_unequal\_split, text = **"No "**, variable = UNEQUAL\_SPLIT, value = False, font = (**'arial'**, 10)).pack(side = LEFT)  
 rg\_dual\_type=Frame(Forms)  
 Yes=Radiobutton(rg\_dual\_type, text = **"Yes "**, variable = DUAL\_TYPE, value = True, font = (**'arial'**, 10)).pack(side = LEFT)  
 No=Radiobutton(rg\_dual\_type, text = **"No "**, variable = DUAL\_TYPE, value = False, font = (**'arial'**, 10)).pack(side = LEFT)  
 rg\_lab\_vr=Frame(Forms)  
 Yes=Radiobutton(rg\_lab\_vr, text = **"Yes "**, variable = LAB\_VR, value = True, font = (**'arial'**, 10)).pack(side = LEFT)  
 No=Radiobutton(rg\_lab\_vr, text = **"No "**, variable = LAB\_VR, value = False, font = (**'arial'**, 10)).pack(side = LEFT)  
 rg\_xdept\_room\_utility=Frame(Forms)  
 Yes=Radiobutton(rg\_xdept\_room\_utility, text = **"Yes "**, variable = XDEPT\_ROOM\_UTILITY, value = True, font = (**'arial'**, 10)).pack(side = LEFT)  
 No=Radiobutton(rg\_xdept\_room\_utility, text = **"No "**, variable = XDEPT\_ROOM\_UTILITY, value = False, font = (**'arial'**, 10)).pack(side = LEFT)  
 rg\_instructor\_overload=Frame(Forms)  
 Yes=Radiobutton(rg\_instructor\_overload, text = **"Yes "**, variable = INSTRUCTOR\_OVERLOAD, value = True, font = (**'arial'**, 10)).pack(side = LEFT)  
 No=Radiobutton(rg\_instructor\_overload, text = **"No "**, variable = INSTRUCTOR\_OVERLOAD, value = False, font = (**'arial'**, 10)).pack(side = LEFT)  
 rg\_instructor\_availability=Frame(Forms)  
 Yes=Radiobutton(rg\_instructor\_availability, text = **"Yes "**, variable = INSTRUCTOR\_AVAILABILITY, value = True, font = (**'arial'**, 10)).pack(side = LEFT)  
 No=Radiobutton(rg\_instructor\_availability, text = **"No "**, variable = INSTRUCTOR\_AVAILABILITY, value = False, font = (**'arial'**, 10)).pack(side = LEFT)  
 rg\_room\_availability=Frame(Forms)  
 Yes=Radiobutton(rg\_room\_availability, text = **"Yes "**, variable = ROOM\_AVAILABILITY, value = True, font = (**'arial'**, 10)).pack(side = LEFT)  
 No=Radiobutton(rg\_room\_availability, text = **"No "**, variable = ROOM\_AVAILABILITY, value = False, font = (**'arial'**, 10)).pack(side = LEFT)  
 rg\_section\_availability=Frame(Forms)  
 Yes=Radiobutton(rg\_section\_availability, text = **"Yes "**, variable = SECTION\_AVAILABILITY, value = True, font = (**'arial'**, 10)).pack(side = LEFT)  
 No=Radiobutton(rg\_section\_availability, text = **"No "**, variable = SECTION\_AVAILABILITY, value = False, font = (**'arial'**, 10)).pack(side = LEFT)  
  
 *# ==================================LABEL WIDGET=======================================* txt\_title=Label(Top, width = 900, font = (**'arial'**, 24), text = **"Settings"**, fg = **"White"**,  
 bg = **"#c31d3a"**) *# Green = #008000, REd = #FF0000 Magenta = #FF00FF* txt\_title.pack()  
 txt\_population\_size=Label(Forms, text = **"Population Size:"**, font = (**'arial'**, 10), bd = 15)  
 txt\_population\_size.grid(row = 0, stick = **"e"**)  
 txt\_mutation\_rate=Label(Forms, text = **"Mutation Rate (decimal):"**, font = (**'arial'**, 10), bd = 15)  
 txt\_mutation\_rate.grid(row = 1, stick = **"e"**)  
 txt\_number\_of\_trials=Label(Forms, text = **"Number of Trials:"**, font = (**'arial'**, 10), bd = 15)  
 txt\_number\_of\_trials.grid(row = 2, stick = **"e"**)  
 txt\_allowable\_room\_conflicts=Label(Forms, text = **"Allowable Room Conflicts % (decimal):"**, font = (**'arial'**, 10), bd = 15)  
 txt\_allowable\_room\_conflicts.grid(row = 3, stick = **"e"**)  
 txt\_consecutive\_teaching\_hours=Label(Forms, text = **"Consecutive Teaching Hours:"**, font = (**'arial'**, 10), bd = 15)  
 txt\_consecutive\_teaching\_hours.grid(row = 4, stick = **"e"**)  
 txt\_scheduling\_time\_limit=Label(Forms, text = **"Scheduling Time Limit (hours):"**, font = (**'arial'**, 10), bd = 15)  
 txt\_scheduling\_time\_limit.grid(row = 5, stick = **"e"**)  
 txt\_recurring\_update\_time=Label(Forms, text = **"Recuring Update Time (hours):"**, font = (**'arial'**, 10), bd = 15)  
 txt\_recurring\_update\_time.grid(row = 6, stick = **"e"**)  
 txt\_verbose\_flag=Label(Forms, text = **"Enable Verbose Mode?"**, font = (**'arial'**, 10), bd = 15)  
 txt\_verbose\_flag.grid(row = 7, stick = **"e"**)  
 txt\_max\_room\_limit=Label(Forms, text = **"Enable Max Room Limit?"**, font = (**'arial'**, 10), bd = 15)  
 txt\_max\_room\_limit.grid(row = 8, stick = **"e"**)  
 txt\_casual\_split=Label(Forms, text = **"Disable Same Day Time Split?"**, font = (**'arial'**, 10), bd = 15)  
 txt\_casual\_split.grid(row = 9, stick = **"e"**)  
 txt\_unequal\_split=Label(Forms, text = **"Disable Unequal Time Split?"**, font = (**'arial'**, 10), bd = 15)  
 txt\_unequal\_split.grid(row = 10, stick = **"e"**)  
 txt\_dual\_type=Label(Forms, text = **"Disable Dual Class Type?"**, font = (**'arial'**, 10), bd = 15)  
 txt\_dual\_type.grid(row = 11, stick = **"e"**)  
 txt\_lab\_vr=Label(Forms, text = **"Allow Lab Subjects in Virtual Room?"**, font = (**'arial'**, 10), bd = 15)  
 txt\_lab\_vr.grid(row = 12, stick = **"e"**)  
 txt\_xdept\_room\_utility=Label(Forms, text = **"Disabe XDept Room Utility?"**, font = (**'arial'**, 10), bd = 15)  
 txt\_xdept\_room\_utility.grid(row = 13, stick = **"e"**)  
 txt\_instructor\_overload=Label(Forms, text = **"Disable Instructor Overload?"**, font = (**'arial'**, 10), bd = 15)  
 txt\_instructor\_overload.grid(row = 14, stick = **"e"**)  
 txt\_instructor\_availability=Label(Forms, text = **"Enable Instructor Availability?"**, font = (**'arial'**, 10), bd = 15)  
 txt\_instructor\_availability.grid(row = 15, stick = **"e"**)  
 txt\_room\_availability=Label(Forms, text = **"Enable Room Availability?"**, font = (**'arial'**, 10), bd = 15)  
 txt\_room\_availability.grid(row = 16, stick = **"e"**)  
 txt\_section\_availability=Label(Forms, text = **"Enable Section Availability?"**, font = (**'arial'**, 10), bd = 15)  
 txt\_section\_availability.grid(row = 17, stick = **"e"**)  
 txt\_result=Label(Buttons)  
 txt\_result.pack(side = TOP)  
  
 *# ==================================ENTRY WIDGET=======================================* e\_population\_size=Entry(Forms, textvariable = POPULATION\_SIZE, width = 15)  
 e\_population\_size.grid(row = 0, column = 1)  
 e\_mutation\_rate=Entry(Forms, textvariable = MUTATION\_RATE, width = 15)  
 e\_mutation\_rate.grid(row = 1, column = 1)  
 e\_number\_of\_trials=Entry(Forms, textvariable = NUMBER\_OF\_TRIALS, width = 15)  
 e\_number\_of\_trials.grid(row = 2, column = 1)  
 e\_number\_of\_trials.config(state = **"disabled"**) *# Delete this to enable the entry box again if the bug on this feature is solved.* e\_allowable\_room\_conflicts=Entry(Forms, textvariable = ALLOWABLE\_ROOM\_CONFLICTS, width = 15)  
 e\_allowable\_room\_conflicts.grid(row = 3, column = 1)  
 e\_consecutive\_teaching\_hours=Entry(Forms, textvariable = CONSECUTIVE\_TEACHING\_HOURS, width = 15)  
 e\_consecutive\_teaching\_hours.grid(row = 4, column = 1)  
 e\_scheduling\_time\_limit=Entry(Forms, textvariable = SCHEDULING\_TIME\_LIMIT, width = 15)  
 e\_scheduling\_time\_limit.grid(row = 5, column = 1)  
 e\_recurring\_update\_time=Entry(Forms, textvariable = RECURRING\_UPDATE\_TIME, width = 15)  
 e\_recurring\_update\_time.grid(row = 6, column = 1)  
 rg\_verbose\_flag.grid(row = 7, column = 1)  
 rg\_max\_room\_limit.grid(row = 8, column = 1)  
 rg\_casual\_split.grid(row = 9, column = 1)  
 rg\_unequal\_split.grid(row = 10, column = 1)  
 rg\_dual\_type.grid(row = 11, column = 1)  
 rg\_lab\_vr.grid(row = 12, column = 1)  
 rg\_xdept\_room\_utility.grid(row = 13, column = 1)  
 rg\_instructor\_overload.grid(row = 14, column = 1)  
 rg\_instructor\_availability.grid(row = 15, column = 1)  
 rg\_room\_availability.grid(row = 16, column = 1)  
 rg\_section\_availability.grid(row = 17, column = 1)  
  
 *# ==================================BUTTONS WIDGET=====================================* btn\_save=Button(Buttons, width = 55, text = **"Save"**, font = (**'arial'**, 13, **'bold'**), command = Save)  
 btn\_save.pack(side = LEFT)  
  
 *# ==================================INITIALIZATION=====================================* if \_\_name\_\_ == **'\_\_main\_\_'**:  
 View()  
 setroot.mainloop()  
  
  
 def Outputs():  
 outroot=Toplevel(root)  
 outroot.title(**"ECE-SPV"**)  
 outroot.wm\_iconbitmap(**"ece-spv.ico"**)  
 screen\_width=outroot.winfo\_screenwidth()  
 screen\_height=outroot.winfo\_screenheight()  
 width=400  
 height=960  
 x=(screen\_width / 2) - (width / 2)  
 y=(screen\_height / 2) - (height / 2)  
 outroot.geometry(**'%dx%d+%d+%d'** % (width, height, x, y))  
 outroot.resizable(0, 0)  
  
 *# ==================================COMMANDS============================================* def Save():  
 *# writing new config* with open(OUTPUT\_INFO, **'w'**) as f:  
 f.write(**'SCHOOL\_NAME='** + str(SCHOOL\_NAME.get()) + **'**\n**'** +  
 **'TOP\_HEADER='** + str(TOP\_HEADER.get()) + **'**\n**'** +  
 **'MID\_HEADER='** + str(MID\_HEADER.get()) + **'**\n**'** +  
 **'BOT\_HEADER='** + str(BOT\_HEADER.get()) + **'**\n**'** +  
 **'UNDERLOGO\_TEXT='** + str(UNDERLOGO\_TEXT.get()) + **'**\n**'** +  
 **'COLLEGE='** + str(COLLEGE.get()) + **'**\n**'** +  
 **'DEPARTMENT='** + str(DEPARTMENT.get()) + **'**\n**'** +  
 **'DEPARTMENT\_HEAD='** + str(DEPARTMENT\_HEAD.get()) + **'**\n**'** +  
 **'DEAN='** + str(DEAN.get()) + **'**\n**'** +  
 **'INDEX\_NO='** + str(INDEX\_NO.get()) + **'**\n**'** +  
 **'ISSUE\_NO='** + str(ISSUE\_NO.get()) + **'**\n**'** +  
 **'REVISION\_NO='** + str(REV\_NO.get()) + **'**\n**'** +  
 **'PAGE\_NO='** + str(PAGE\_NO.get()) + **'**\n**'** +  
 **'QAC\_NO='** + str(QAC\_NO.get()) + **'**\n**'** +  
 **'DATE='** + str(DATE.get()) + **'**\n**'** +  
 **'SCHOOL\_YEAR='** + str(SCHOOL\_YEAR.get()) + **'**\n**'** +  
 **'SEMESTER='** + str(SEMESTER.get()))  
 f.close()  
 txt\_result.config(text = **"The changes are saved!"**, fg = **"green"**)  
  
 *# ==================================VARIABLES==========================================* SCHOOL\_NAME=StringVar(outroot)  
 TOP\_HEADER=StringVar(outroot)  
 MID\_HEADER=StringVar(outroot)  
 BOT\_HEADER=StringVar(outroot)  
 UNDERLOGO\_TEXT=StringVar(outroot)  
 COLLEGE=StringVar(outroot)  
 DEPARTMENT=StringVar(outroot)  
 DEPARTMENT\_HEAD=StringVar(outroot)  
 DEAN=StringVar(outroot)  
 INDEX\_NO=StringVar(outroot)  
 ISSUE\_NO=StringVar(outroot)  
 REV\_NO=StringVar(outroot)  
 PAGE\_NO=StringVar(outroot)  
 QAC\_NO=StringVar(outroot)  
 DATE=StringVar(outroot)  
 SCHOOL\_YEAR=StringVar(outroot)  
 SEMESTER=StringVar(outroot)  
  
 try:  
 with open(OUTPUT\_INFO, **'r'**) as f:  
 lines=f.readlines()  
 SCHOOL\_NAME.set(str(lines[0].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 TOP\_HEADER.set(str(lines[1].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 MID\_HEADER.set(str(lines[2].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 BOT\_HEADER.set(str(lines[3].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 UNDERLOGO\_TEXT.set(str(lines[4].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 COLLEGE.set(str(lines[5].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 DEPARTMENT.set(str(lines[6].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 DEPARTMENT\_HEAD.set(str(lines[7].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 DEAN.set(str(lines[8].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 INDEX\_NO.set(str(lines[9].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 ISSUE\_NO.set(str(lines[10].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 REV\_NO.set(str(lines[11].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 PAGE\_NO.set(str(lines[12].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 QAC\_NO.set(str(lines[13].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 DATE.set(str(lines[14].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 SCHOOL\_YEAR.set(str(lines[15].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 SEMESTER.set(str(lines[16].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 f.close()  
 except IOError: *# this is what happens if the file doesn't exist* with open(OUTPUT\_INFO, **'w'**) as f:  
 d\_SCHOOL\_NAME=**''** d\_TOP\_HEADER=**''** d\_MID\_HEADER=**''** d\_BOT\_HEADER=**''** d\_UNDERLOGO\_TEXT=**''** d\_COLLEGE=**''** d\_DEPARTMENT=**''** d\_DEPARTMENT\_HEAD=**''** d\_DEAN=**''** d\_INDEX\_NO=**''** d\_ISSUE\_NO=**''** d\_REV\_NO=**''** d\_PAGE\_NO=**''** d\_QAC\_NO=**''** d\_DATE=**''** d\_SCHOOL\_YEAR=**''** d\_SEMESTER=**''** f.write(**'SCHOOL\_NAME='** + str(d\_SCHOOL\_NAME) + **'**\n**'** +  
 **'TOP\_HEADER='** + str(d\_TOP\_HEADER) + **'**\n**'** +  
 **'MID\_HEADER='** + str(d\_MID\_HEADER) + **'**\n**'** +  
 **'BOT\_HEADER='** + str(d\_BOT\_HEADER) + **'**\n**'** +  
 **'UNDERLOGO\_TEXT='** + str(d\_UNDERLOGO\_TEXT) + **'**\n**'** +  
 **'COLLEGE='** + str(d\_COLLEGE) + **'**\n**'** +  
 **'DEPARTMENT='** + str(d\_DEPARTMENT) + **'**\n**'** +  
 **'DEPARTMENT\_HEAD='** + str(d\_DEPARTMENT\_HEAD) + **'**\n**'** +  
 **'DEAN='** + str(d\_DEAN) + **'**\n**'** +  
 **'INDEX\_NO='** + str(d\_INDEX\_NO) + **'**\n**'** +  
 **'ISSUE\_NO='** + str(d\_ISSUE\_NO) + **'**\n**'** +  
 **'REVISION\_NO='** + str(d\_REV\_NO) + **'**\n**'** +  
 **'PAGE\_NO='** + str(d\_PAGE\_NO) + **'**\n**'** +  
 **'QAC\_NO='** + str(d\_QAC\_NO) + **'**\n**'** +  
 **'DATE='** + str(d\_DATE) + **'**\n**'** +  
 **'SCHOOL\_YEAR='** + str(d\_SCHOOL\_YEAR) + **'**\n**'** +  
 **'SEMESTER='** + str(d\_SEMESTER))  
 f.close()  
  
 *# ==================================FRAME==============================================* Top=Frame(outroot, width = 900, height = 50, bd = 8, relief = **"raise"**, bg = **"#292bc2"**)  
 Top.pack(side = TOP)  
 Left=Frame(outroot, width = 300, height = 500, bd = 8, relief = **"raise"**)  
 Left.pack(side = LEFT)  
 Forms=Frame(Left, width = 300, height = 450)  
 Forms.pack(side = TOP)  
 Buttons=Frame(Left, width = 300, height = 100, bd = 8, relief = **"raise"**)  
 Buttons.pack(side = BOTTOM)  
  
 *# ==================================LABEL WIDGET=======================================* txt\_title=Label(Top, width = 900, font = (**'arial'**, 24), text = **"Outputs"**, fg = **"White"**,  
 bg = **"#3639ff"**)   
 txt\_title.pack()  
 txt\_school\_name=Label(Forms, text = **"School Name:"**, font = (**'arial'**, 10), bd = 15)  
 txt\_school\_name.grid(row = 0, stick = **"e"**)  
 txt\_top\_header=Label(Forms, text = **"Top Header:"**, font = (**'arial'**, 10), bd = 15)  
 txt\_top\_header.grid(row = 1, stick = **"e"**)  
 txt\_mid\_header=Label(Forms, text = **"Mid Header:"**, font = (**'arial'**, 10), bd = 15)  
 txt\_mid\_header.grid(row = 2, stick = **"e"**)  
 txt\_bot\_header=Label(Forms, text = **"Bot Header:"**, font = (**'arial'**, 10), bd = 15)  
 txt\_bot\_header.grid(row = 3, stick = **"e"**)  
 txt\_underlogo\_text=Label(Forms, text = **"Underlogo Text:"**, font = (**'arial'**, 10), bd = 15)  
 txt\_underlogo\_text.grid(row = 4, stick = **"e"**)  
 txt\_college=Label(Forms, text = **"College:"**, font = (**'arial'**, 10), bd = 15)  
 txt\_college.grid(row = 5, stick = **"e"**)  
 txt\_department=Label(Forms, text = **"Department:"**, font = (**'arial'**, 10), bd = 15)  
 txt\_department.grid(row = 6, stick = **"e"**)  
 txt\_department\_head=Label(Forms, text = **"Department Head:"**, font = (**'arial'**, 10), bd = 15)  
 txt\_department\_head.grid(row = 7, stick = **"e"**)  
 txt\_dean=Label(Forms, text = **"Dean:"**, font = (**'arial'**, 10), bd = 15)  
 txt\_dean.grid(row = 8, stick = **"e"**)  
 txt\_index\_no=Label(Forms, text = **"Index Number:"**, font = (**'arial'**, 10), bd = 15)  
 txt\_index\_no.grid(row = 9, stick = **"e"**)  
 txt\_issue\_no=Label(Forms, text = **"Issue Number:"**, font = (**'arial'**, 10), bd = 15)  
 txt\_issue\_no.grid(row = 10, stick = **"e"**)  
 txt\_rev\_no=Label(Forms, text = **"Revision Number:"**, font = (**'arial'**, 10), bd = 15)  
 txt\_rev\_no.grid(row = 11, stick = **"e"**)  
 txt\_page\_no=Label(Forms, text = **"Page Number:"**, font = (**'arial'**, 10), bd = 15)  
 txt\_page\_no.grid(row = 12, stick = **"e"**)  
 txt\_qac\_no=Label(Forms, text = **"QAC Number:"**, font = (**'arial'**, 10), bd = 15)  
 txt\_qac\_no.grid(row = 13, stick = **"e"**)  
 txt\_date=Label(Forms, text = **"Date:"**, font = (**'arial'**, 10), bd = 15)  
 txt\_date.grid(row = 14, stick = **"e"**)  
 txt\_school\_year=Label(Forms, text = **"School Year:"**, font = (**'arial'**, 10), bd = 15)  
 txt\_school\_year.grid(row = 15, stick = **"e"**)  
 txt\_semester=Label(Forms, text = **"Semester:"**, font = (**'arial'**, 10), bd = 15)  
 txt\_semester.grid(row = 16, stick = **"e"**)  
 txt\_result=Label(Buttons)  
 txt\_result.pack(side = TOP)  
  
 *# ==================================ENTRY WIDGET=======================================* e\_school\_name=Entry(Forms, textvariable = SCHOOL\_NAME, width = 36)  
 e\_school\_name.grid(row = 0, column = 1)  
 e\_top\_header=Entry(Forms, textvariable = TOP\_HEADER, width = 36)  
 e\_top\_header.grid(row = 1, column = 1)  
 e\_mid\_header=Entry(Forms, textvariable = MID\_HEADER, width = 36)  
 e\_mid\_header.grid(row = 2, column = 1)  
 e\_bot\_header=Entry(Forms, textvariable = BOT\_HEADER, width = 36)  
 e\_bot\_header.grid(row = 3, column = 1)  
 e\_underlogo\_text=Entry(Forms, textvariable = UNDERLOGO\_TEXT, width = 36)  
 e\_underlogo\_text.grid(row = 4, column = 1)  
 e\_college=Entry(Forms, textvariable = COLLEGE, width = 36)  
 e\_college.grid(row = 5, column = 1)  
 e\_department=Entry(Forms, textvariable = DEPARTMENT, width = 36)  
 e\_department.grid(row = 6, column = 1)  
 e\_department\_head=Entry(Forms, textvariable = DEPARTMENT\_HEAD, width = 36)  
 e\_department\_head.grid(row = 7, column = 1)  
 e\_dean=Entry(Forms, textvariable = DEAN, width = 36)  
 e\_dean.grid(row = 8, column = 1)  
 e\_index\_no=Entry(Forms, textvariable = INDEX\_NO, width = 36)  
 e\_index\_no.grid(row = 9, column = 1)  
 e\_issue\_no=Entry(Forms, textvariable = ISSUE\_NO, width = 36)  
 e\_issue\_no.grid(row = 10, column = 1)  
 e\_rev\_no=Entry(Forms, textvariable = REV\_NO, width = 36)  
 e\_rev\_no.grid(row = 11, column = 1)  
 e\_page\_no=Entry(Forms, textvariable = PAGE\_NO, width = 36)  
 e\_page\_no.grid(row = 12, column = 1)  
 e\_qac\_no=Entry(Forms, textvariable = QAC\_NO, width = 36)  
 e\_qac\_no.grid(row = 13, column = 1)  
 e\_date=Entry(Forms, textvariable = DATE, width = 36)  
 e\_date.grid(row = 14, column = 1)  
 e\_school\_year=Entry(Forms, textvariable = SCHOOL\_YEAR, width = 36)  
 e\_school\_year.grid(row = 15, column = 1)  
 e\_semester=Entry(Forms, textvariable = SEMESTER, width = 36)  
 e\_semester.grid(row = 16, column = 1)  
  
 *# ==================================BUTTONS WIDGET=====================================* btn\_save=Button(Buttons, width = 55, text = **"Save"**, font = (**'arial'**, 13, **'bold'**), command = Save)  
 btn\_save.pack(side = LEFT)  
  
 *# ==================================INITIALIZATION=====================================* if \_\_name\_\_ == **'\_\_main\_\_'**:  
 View()  
 outroot.mainloop()  
  
  
 def f():  
 btn\_find[**'state'**] = DISABLED  
  
 txt\_result.config(text = **"Find Optimized Schedule Button Clicked!"**, fg = **"#EE7600"**)  
  
 with open(SETTING\_INFO, **'r'**) as f:  
 def str2bool(v):  
 return v.lower() in (**"True"**, **"true"**, **"Yes"**, **"yes"**)  
  
 lines=f.readlines()  
 POPULATION\_SIZE=(int(lines[0].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 MUTATION\_RATE=(float(lines[1].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 NUMBER\_OF\_TRIALS=(int(lines[2].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 ALLOWABLE\_ROOM\_CONFLICTS=(float(lines[3].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 CONSECUTIVE\_TEACHING\_HOURS=(float(lines[4].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 SCHEDULING\_TIME\_LIMIT=(float(lines[5].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 RECURRING\_UPDATE\_TIME=(float(lines[6].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 VERBOSE\_FLAG=(str2bool(lines[7].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 ENABLE\_NUMB\_OF\_STUDENTS=(str2bool(lines[8].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 DISABLE\_CASUAL\_SPLITTING=(str2bool(lines[9].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 DISABLE\_UNEQUAL\_SPLIT=(str2bool(lines[10].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 DISABLE\_MIXED\_TYPE=(str2bool(lines[11].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 LAB\_ON\_VR=(str2bool(lines[12].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 DISABLE\_XDEPT\_ROOM\_UTILITY=(str2bool(lines[13].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 DISABLE\_INSTRUCTOR\_OVERLOAD=(str2bool(lines[14].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 LIMIT\_INSTRUCTOR\_AVAILABILITY=(str2bool(lines[15].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 LIMIT\_ROOM\_AVAILABILITY=(str2bool(lines[16].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 LIMIT\_SECTION\_AVAILABILITY=(str2bool(lines[17].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 f.close()  
  
 with open(OUTPUT\_INFO, **'r'**) as f:  
 lines=f.readlines()  
 schooltext=(str(lines[0].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 school\_infotext=(str(lines[1].split(**'='**, 1)[1].rstrip(**'**\n**'**)) + **'**\n**'** + str(lines[2].split(**'='**, 1)[1].rstrip(**'**\n**'**)) + **'**\n**'** + str(lines[3].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 vaa\_oap=(str(lines[4].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 college=(str(lines[5].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 department=(str(lines[6].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 depthead=(str(lines[7].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 dean=(str(lines[8].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 indexno=(str(lines[9].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 issueno=(str(lines[10].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 revisionno=(str(lines[11].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 page=(str(lines[12].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 qacno=(str(lines[13].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 date=(str(lines[14].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 schoolyear=(str(lines[15].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 semester=(str(lines[16].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 f.close()  
  
 NUMB\_OF\_ELITE\_SCHEDULES=1  
 TOURNAMENT\_SELECTION\_SIZE=3  
  
 class DBMgr:  
 def \_\_init\_\_(self):  
 self.\_conn=sqlite.connect(**'ARADB.db'**) *# sql connector* self.\_cursor=self.\_conn.cursor() *# sql cursor* self.\_times=self.\_select\_times() *# select time sql cursor* self.\_timepoints=self.\_select\_timepoints() *# select timepoints sql cursor* self.\_meetingTimes=self.\_select\_meeting\_times() *# select meeting time sql cursor* self.\_instructors=self.\_select\_instructors() *# select instructor ql cursor* self.\_subjects=self.\_select\_subjects() *# select subject sql cursor* self.\_rooms=self.\_select\_rooms() *# select room sql cursor* self.\_sections=self.\_select\_sections() *# select section sql cursor* self.\_numberOfClasses=0 *# initial number of classes* for i in range(0, len(self.\_sections)):  
 self.\_numberOfClasses+=len(  
 self.\_sections[i].get\_subjects()) *# auto-itereation of number of classes creation  
  
 # select section sql command* def \_select\_sections(self):  
 self.\_cursor.execute(**"SELECT \* FROM section"**)  
 sections=self.\_cursor.fetchall()  
 returnSections=[]  
 for i in range(0, len(sections)):  
 returnSections.append(Section(sections[i][0], self.\_select\_section\_subjects(sections[i][0]),  
 self.\_select\_section\_availability(sections[i][0])))  
 return returnSections  
  
 *# select subject sql command* def \_select\_subjects(self):  
 self.\_cursor.execute(**"SELECT \* FROM subject"**)  
 subjects=self.\_cursor.fetchall()  
 returnSubjects=[]  
 for i in range(0, len(subjects)):  
 returnSubjects.append(  
 Subject(subjects[i][0], subjects[i][1], subjects[i][2], subjects[i][3], subjects[i][4],  
 subjects[i][5], subjects[i][6], subjects[i][7], self.\_select\_subject\_instructors(  
 subjects[i][0])))   
 return returnSubjects  
  
 *# select instructor sql command* def \_select\_instructors(self):  
 self.\_cursor.execute(**"SELECT \* FROM instructor"**)  
 instructors=self.\_cursor.fetchall()  
 returnInstructors=[]  
 for i in range(0, len(instructors)):  
 returnInstructors.append(  
 Instructor(instructors[i][0], instructors[i][1], instructors[i][2], instructors[i][3],  
 self.\_select\_instructor\_availability(instructors[i][0]),  
 self.\_select\_instructor\_availability\_start(instructors[i][0]),  
 self.\_select\_instructor\_availability\_end(instructors[i][0]),  
 self.\_select\_instructor\_overtime(instructors[i][0])))  
 return returnInstructors  
  
 *# select room sql command* def \_select\_rooms(self):  
 self.\_cursor.execute(**"SELECT \* FROM room"**)  
 rooms=self.\_cursor.fetchall()  
 returnRooms=[]  
 for i in range(0, len(rooms)):  
 returnRooms.append(Room(rooms[i][0], rooms[i][1], rooms[i][2], rooms[i][3],  
 self.\_select\_room\_availability(  
 rooms[i][  
 0])))   
 return returnRooms  
  
 *# select meeting time sql command* def \_select\_meeting\_times(self):  
 self.\_cursor.execute(**"SELECT \* FROM meeting\_time"**)  
 meetingTimes=self.\_cursor.fetchall()  
 returnMeetingTimes=[]  
 for i in range(0, len(meetingTimes)):  
 returnMeetingTimes.append(MeetingTime(meetingTimes[i][0], meetingTimes[i][1], meetingTimes[i][2],  
 meetingTimes[i][3], meetingTimes[i][4],  
 self.\_select\_rest\_time(meetingTimes[i][0]),  
 self.\_select\_break\_time(meetingTimes[i][0])))   
 return returnMeetingTimes  
  
 *# select time sql command* def \_select\_times(self):  
 self.\_cursor.execute(**"SELECT \* FROM time"**)  
 times=self.\_cursor.fetchall()  
 returnTimes=[]  
 for i in range(0, len(times)):  
 returnTimes.append(Time(times[i][0]))  
 return returnTimes  
  
 *# select timepoint sql command* def \_select\_timepoints(self):  
 self.\_cursor.execute(**"SELECT \* FROM time\_point"**)  
 timepoints=self.\_cursor.fetchall()  
 returnTimePoints=[]  
 for i in range(0, len(timepoints)):  
 returnTimePoints.append(TimePoint(timepoints[i][0],  
 self.\_select\_point\_block(timepoints[i][0]),  
 self.\_select\_phantom\_block(timepoints[i][0])))  
 return returnTimePoints  
  
 *# GETTING VALUE* def \_select\_section\_subjects(self, sectionCode):  
 self.\_cursor.execute(**"SELECT \* FROM section\_subject where section\_code == '"** + sectionCode + **"'"**)  
 dbSubjectCodes=self.\_cursor.fetchall()  
 subjectCodes=[]  
 for i in range(0, len(dbSubjectCodes)):  
 subjectCodes.append(dbSubjectCodes[i][1])  
 returnValue=[]  
 for i in range(0, len(self.\_subjects)):  
 if self.\_subjects[i].get\_code() in subjectCodes:  
 returnValue.append(self.\_subjects[i])  
 return returnValue  
  
 def \_select\_subject\_instructors(self, subjectCode):  
 self.\_cursor.execute(**"SELECT \* FROM subject\_instructor where subject\_code == '"** + subjectCode + **"'"**)  
 dbInstructorNumbers=self.\_cursor.fetchall()  
 instructorNumbers=[]  
 for i in range(0, len(dbInstructorNumbers)): instructorNumbers.append(dbInstructorNumbers[i][1])  
 returnValue=[]  
 for i in range(0, len(self.\_instructors)):  
 if self.\_instructors[i].get\_number() in instructorNumbers:  
 returnValue.append(self.\_instructors[i])  
 return returnValue  
  
 def \_select\_instructor\_availability(self, instructor):  
 self.\_cursor.execute(  
 **"SELECT \* from instructor\_availability where instructor\_id == '"** + instructor + **"'"**)  
 instructorMTsRS=self.\_cursor.fetchall()  
 instructorMTs=[]  
 for i in range(0, len(instructorMTsRS)):  
 instructorMTs.append(instructorMTsRS[i][1])  
 instructorAvailability=list()  
 for i in range(0, len(self.\_meetingTimes)):  
 if self.\_meetingTimes[i].get\_id() in instructorMTs:  
 instructorAvailability.append(self.\_meetingTimes[i].get\_id())  
 return instructorAvailability  
  
 def \_select\_instructor\_availability\_start(self, instructor):  
 self.\_cursor.execute(  
 **"SELECT \* from instructor\_availability where instructor\_id == '"** + instructor + **"'"**)  
 instructorMTsRS=self.\_cursor.fetchall()  
 instructorMTsSTART=[]  
 for i in range(0, len(instructorMTsRS)):  
 instructorMTsSTART.append(instructorMTsRS[i][2])  
 instructorAvailabilitySTART=list()  
 for i in range(0, len(self.\_timepoints)):  
 if self.\_timepoints[i].get\_point() in instructorMTsSTART:  
 instructorAvailabilitySTART.append(self.\_timepoints[i].get\_point())  
 return instructorAvailabilitySTART  
  
 def \_select\_instructor\_availability\_end(self, instructor):  
 self.\_cursor.execute(  
 **"SELECT \* from instructor\_availability where instructor\_id == '"** + instructor + **"'"**)  
 instructorMTsRS=self.\_cursor.fetchall()  
 instructorMTsEND=[]  
 for i in range(0, len(instructorMTsRS)):  
 instructorMTsEND.append(instructorMTsRS[i][3])  
 instructorAvailabilityEND=list()  
 for i in range(0, len(self.\_timepoints)):  
 if self.\_timepoints[i].get\_point() in instructorMTsEND:  
 instructorAvailabilityEND.append(self.\_timepoints[i].get\_point())  
 return instructorAvailabilityEND  
  
 def \_select\_instructor\_overtime(self, instructor):  
 self.\_cursor.execute(**"SELECT \* from instructor\_overtime where instructor\_id == '"** + instructor + **"'"**)  
 instructorMTsRS=self.\_cursor.fetchall()  
 instructorMTs=[]  
 for i in range(0, len(instructorMTsRS)):  
 instructorMTs.append(instructorMTsRS[i][1])  
 instructorOvertime=list()  
 for i in range(0, len(self.\_meetingTimes)):  
 if self.\_meetingTimes[i].get\_id() in instructorMTs:  
 instructorOvertime.append(self.\_meetingTimes[i].get\_id())  
 return instructorOvertime  
  
 def \_select\_room\_availability(self, room):  
 self.\_cursor.execute(**"SELECT \* from room\_availability where room\_number = '"** + room + **"'"**)  
 roomMTsRS=self.\_cursor.fetchall()  
 roomMTs=[]  
 for i in range(0, len(roomMTsRS)): roomMTs.append(roomMTsRS[i][1])  
 roomAvailability=list()  
 for i in range(0, len(self.\_meetingTimes)):  
 if self.\_meetingTimes[i].get\_id() in roomMTs:  
 roomAvailability.append(self.\_meetingTimes[i].get\_id())  
 return roomAvailability  
  
 def \_select\_section\_availability(self, section):  
 self.\_cursor.execute(**"SELECT \* from section\_availability where section\_code = '"** + section + **"'"**)  
 sectionMTsRS=self.\_cursor.fetchall()  
 sectionMTs=[]  
 for i in range(0, len(sectionMTsRS)): sectionMTs.append(sectionMTsRS[i][1])  
 sectionAvailability=list()  
 for i in range(0, len(self.\_meetingTimes)):  
 if self.\_meetingTimes[i].get\_id() in sectionMTs:  
 sectionAvailability.append(self.\_meetingTimes[i].get\_id())  
 return sectionAvailability  
  
 def \_select\_rest\_time(self, meeting\_time):  
 self.\_cursor.execute(**"SELECT \* from rest\_time where meeting\_time\_id = '"** + meeting\_time + **"'"**)  
 timeMTsRS=self.\_cursor.fetchall()  
 timeMTs=[]  
 for i in range(0, len(timeMTsRS)): timeMTs.append(timeMTsRS[i][1])  
 restTime=list()  
 for i in range(0, len(self.\_times)):  
 if self.\_times[i].get\_block() in timeMTs:  
 restTime.append(self.\_times[i].get\_block())  
 return restTime  
  
 def \_select\_break\_time(self, meeting\_time):  
 self.\_cursor.execute(**"SELECT \* from break\_time where meeting\_time\_id = '"** + meeting\_time + **"'"**)  
 timeMTsRS=self.\_cursor.fetchall()  
 timeMTs=[]  
 for i in range(0, len(timeMTsRS)): timeMTs.append(timeMTsRS[i][1])  
 breakTime=list()  
 for i in range(0, len(self.\_times)):  
 if self.\_times[i].get\_block() in timeMTs:  
 breakTime.append(self.\_times[i].get\_block())  
 return breakTime  
  
 def \_select\_point\_block(self, point):  
 self.\_cursor.execute(**"SELECT \* from point\_block where time\_point == '"** + point + **"'"**)  
 pointBlocksRS=self.\_cursor.fetchall()  
 pointBlocks=[]  
 for i in range(0, len(pointBlocksRS)): pointBlocks.append(pointBlocksRS[i][1])  
 returnValue=list()  
 for i in range(0, len(self.\_times)):  
 if self.\_times[i].get\_block() in pointBlocks:  
 returnValue.append(self.\_times[i].get\_block())  
 return returnValue  
  
 def \_select\_phantom\_block(self, point):  
 self.\_cursor.execute(**"SELECT \* from phantom\_block where time\_point == '"** + point + **"'"**)  
 pointBlocksRS=self.\_cursor.fetchall()  
 pointBlocks=[]  
 for i in range(0, len(pointBlocksRS)): pointBlocks.append(pointBlocksRS[i][1])  
 returnValue=list()  
 for i in range(0, len(self.\_times)):  
 if self.\_times[i].get\_block() in pointBlocks:  
 returnValue.append(self.\_times[i].get\_block())  
 return returnValue  
  
 def get\_sections(self):  
 return self.\_sections *# get item to section table* def get\_subjects(self):  
 return self.\_subjects *# get item to subject table* def get\_instructors(self):  
 return self.\_instructors *# get item to instructor table* def get\_meetingTimes(self):  
 return self.\_meetingTimes *# get item to meeting time table* def get\_times(self):  
 return self.\_times *# get item to time table* def get\_timepoints(self):  
 return self.\_timepoints *# get item to time table* def get\_rooms(self):  
 return self.\_rooms *# get item to room table* def get\_numberOfClasses(self):  
 return self.\_numberOfClasses *# get item to number of class table* class Schedule:  
 def \_\_init\_\_(self):  
 self.\_data=dbMgr  
 self.\_classes=[]  
 self.\_conflicts=[]  
 self.\_fitness=-1  
 self.\_classNumb=0  
 self.\_isFitnessChanged=True  
 self.\_generationNumber=0  
  
 def get\_classes(self):  
 self.\_isFitnessChanged=True  
 return self.\_classes  
  
 def get\_conflicts(self):  
 return self.\_conflicts  
  
 def get\_fitness(self):  
 if (self.\_isFitnessChanged == True):  
 self.\_fitness=self.calculate\_fitness()  
 self.\_isFitnessChanged=False  
 return self.\_fitness  
  
 def initialize(self):  
 sections=dbMgr.get\_sections() *# point to section table* meetingTimes=dbMgr.get\_meetingTimes()  
 rooms=dbMgr.get\_rooms()  
 classes=self.\_classes  
 PR=list()  
 instructorList=list()  
 MT1=list()  
 MT1p5=list()  
 MT2=list()  
 MT3=list()  
 MT4=list()  
 MT1v2=list()  
 MT1p5v2=list()  
 MT2v2=list()  
 MT3v2=list()  
 for RM in range(0, len(rooms)):  
 if (rooms[RM].get\_type() == **'PR'**):  
 PR.append(rooms[RM])  
 for MT in range(0, len(meetingTimes)):  
 if (meetingTimes[MT].get\_MThour() == 1):  
 MT1.append(meetingTimes[MT])  
 MT1v2.append(meetingTimes[MT])  
 if (meetingTimes[MT].get\_MThour() == 1.5):  
 MT1p5.append(meetingTimes[MT])  
 MT1p5v2.append(meetingTimes[MT])  
 if (meetingTimes[MT].get\_MThour() == 2):  
 MT2.append(meetingTimes[MT])  
 MT2v2.append(meetingTimes[MT])  
 if (meetingTimes[MT].get\_MThour() == 3):  
 MT3.append(meetingTimes[MT])  
 MT3v2.append(meetingTimes[MT])  
 if (meetingTimes[MT].get\_MThour() == 4):  
 MT4.append(meetingTimes[MT])  
 for i in range(0, len(sections)): *# List all the items per coloumn* subjects=sections[i].get\_subjects() *# point to subject table* for j in range(0, len(subjects)): *# List all the items per coloumn* newClass=Class(self.\_classNumb, sections[i], subjects[j]) *# Create a new class* self.\_classNumb+=1 *# Iterate the class creation* newClass.set\_instructor(subjects[j].get\_instructors()[rnd.randrange(0, len(subjects[  
 j].get\_instructors()))]) *# Pick a random instructor taht can teach the assigned subject for the new class* if (subjects[j].get\_compatibility() == **'PR'**):  
 newClass.set\_room(  
 PR[rnd.randrange(0, len(PR))]) *# Pick a random room in database for the new class* else:  
 newClass.set\_room(  
 rooms[rnd.randrange(0,  
 len(rooms))]) *# Pick a random room in database for the new class"""* def wholetwohours():  
 newClass.set\_meetingTime(MT2[rnd.randrange(0, len(MT2))])  
 newClass.set\_meetingTime1(dbMgr.get\_meetingTimes()[0])  
  
 def wholethreehours():  
 newClass.set\_meetingTime(MT3[rnd.randrange(0, len(MT3))])  
 newClass.set\_meetingTime1(dbMgr.get\_meetingTimes()[0])  
  
 def wholefourhours():  
 newClass.set\_meetingTime(MT4[rnd.randrange(0, len(MT4))])  
 newClass.set\_meetingTime1(dbMgr.get\_meetingTimes()[0])  
  
 def splittwohours():  
 newClass.set\_meetingTime(MT1[rnd.randrange(0, len(MT1))])  
 newClass.set\_meetingTime1(MT1v2[rnd.randrange(0, len(MT1v2))])  
  
 def splitthreehours():  
 newClass.set\_meetingTime(MT1p5[rnd.randrange(0, len(MT1p5))])  
 newClass.set\_meetingTime1(MT1p5v2[rnd.randrange(0, len(MT1p5v2))])  
  
 def splitfourhours():  
 newClass.set\_meetingTime(MT2[rnd.randrange(0, len(MT2))])  
 newClass.set\_meetingTime1(MT2v2[rnd.randrange(0, len(MT2v2))])  
  
 def call\_funcs\_randomly(funcs):  
 shuffle(funcs)  
 for func in funcs:  
 func()  
  
 *# with mixed Split Time* if (subjects[j].get\_numbHour() == 1):  
 newClass.set\_meetingTime(MT1[rnd.randrange(0, len(MT1))])  
 newClass.set\_meetingTime1(dbMgr.get\_meetingTimes()[0])  
 elif (subjects[j].get\_numbHour() == 2):  
 call\_funcs\_randomly([wholetwohours, splittwohours])  
 elif (subjects[j].get\_numbHour() == 3):  
 call\_funcs\_randomly([wholethreehours, splitthreehours])  
 elif (subjects[j].get\_numbHour() == 4):  
 call\_funcs\_randomly([wholefourhours, splitfourhours])  
 elif (subjects[j].get\_numbHour() == 6):  
 newClass.set\_meetingTime(MT3[rnd.randrange(0, len(MT3))])  
 newClass.set\_meetingTime1(MT3v2[rnd.randrange(0, len(MT3v2))])  
 else:  
 newClass.set\_meetingTime(  
 dbMgr.get\_meetingTimes()[rnd.randrange(0, len(dbMgr.get\_meetingTimes()))])  
 newClass.set\_meetingTime1(  
 dbMgr.get\_meetingTimes()[rnd.randrange(0, len(dbMgr.get\_meetingTimes()))])  
 self.\_classes.append(newClass) *# Add result as new class* return self  
  
 def calculate\_fitness(self):  
 self.\_conflicts=[]  
 classes=self.get\_classes()  
 instructorNames=list()  
 instructor\_Hours=list()  
 for i in range(0, len(classes)):  
 instructorNames.append(classes[i].get\_instructor().get\_name())  
 subjectHours=list()  
 if (classes[i].get\_meetingTime1().get\_id() != **'NULL'**):  
 if (classes[i].get\_meetingTime().get\_id() == classes[i].get\_meetingTime1().get\_id()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(Conflict(Conflict.ConflictType.SAME\_MTS, conflictBetweenClasses))  
 if DISABLE\_UNEQUAL\_SPLIT == True:  
 if (classes[i].get\_meetingTime().get\_MThour() != classes[  
 i].get\_meetingTime1().get\_MThour()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.UNEQUAL\_SPLIT, conflictBetweenClasses))  
 if (classes[i].get\_meetingTime().get\_time() in classes[i].get\_meetingTime1().get\_restTime()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(Conflict(Conflict.ConflictType.OVERLAP\_MTS, conflictBetweenClasses))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[i].get\_meetingTime().get\_restTime()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(Conflict(Conflict.ConflictType.OVERLAP\_MTS, conflictBetweenClasses))  
 if DISABLE\_CASUAL\_SPLITTING == True:  
 *# Disable Same Day Split Time* if (classes[i].get\_meetingTime().get\_day() == classes[i].get\_meetingTime1().get\_day()):  
 if (classes[i].get\_meetingTime().get\_MThour() != classes[  
 i].get\_subject().get\_numbHour()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.CASUAL\_SPLITTING, conflictBetweenClasses))  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[i].get\_meetingTime1().get\_MThour()) !=  
 classes[i].get\_subject().get\_numbHour()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.WRONG\_MTHOUR, conflictBetweenClasses))  
 if ENABLE\_NUMB\_OF\_STUDENTS == True:  
 if (classes[i].get\_room().get\_seatingCapacity() < classes[  
 i].get\_subject().get\_maxNumbOfStudents()):  
 seatingCapacityConflict=list()  
 seatingCapacityConflict.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.NUMB\_OF\_STUDENTS, seatingCapacityConflict))  
 *# Disable to assign Labs on VR* if LAB\_ON\_VR == False:  
 if (classes[i].get\_subject().get\_compatibility() == **'PR'**):  
 if (classes[i].get\_room().get\_type() != **'PR'**):  
 roomCompatibilityConflict=list()  
 roomCompatibilityConflict.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.LAB\_ON\_VR, roomCompatibilityConflict))  
 *# Disable cross department room utility* if DISABLE\_XDEPT\_ROOM\_UTILITY == True:  
 if (classes[i].get\_room().get\_type() != **'VR'**):  
 if (classes[i].get\_subject().get\_dept() != classes[i].get\_room().get\_dept()):  
 roomUtilityConflict=list()  
 roomUtilityConflict.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.XDEPT\_ROOM\_UTILITY, roomUtilityConflict))  
  
 *# if ENABLE\_INSTRUCTOR\_OVERTIME == True:* if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 0* if (classes[i].get\_meetingTime().get\_id() in classes[i].get\_instructor().get\_overtime()): *# This identify the schedules inside the overtime* if (classes[i].get\_meetingTime().get\_id() not in classes[i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_OVERTIME, conflictBetweenClasses))  
 else:  
 *# 0* if (classes[i].get\_meetingTime().get\_id() in classes[i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime().get\_id() not in classes[i].get\_instructor().get\_availability()): *# This removes the schedules out side the official time and overtime* conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_OVERTIME, conflictBetweenClasses))  
 *# 1* if (classes[i].get\_meetingTime1().get\_id() in classes[i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime1().get\_id() not in classes[i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_OVERTIME, conflictBetweenClasses))  
 *# Limit Instructor Availibilty* if LIMIT\_INSTRUCTOR\_AVAILABILITY == True:  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime().get\_id() not in classes[i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_AVAILABILITY, conflictBetweenClasses))  
 else:  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime().get\_id() not in classes[i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_AVAILABILITY, conflictBetweenClasses))  
 *# 1* if (classes[i].get\_meetingTime1().get\_id() not in classes[i].get\_instructor().get\_overtime()):  
 if (classes[i].get\_meetingTime1().get\_id() not in classes[i].get\_instructor().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_AVAILABILITY, conflictBetweenClasses))  
 *# Limit Room Availibilty* if LIMIT\_ROOM\_AVAILABILITY == True:  
 if (classes[i].get\_room().get\_type() != **'VR'**):  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[i].get\_room().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_AVAILABILITY, conflictBetweenClasses))  
 else:  
 *# 1* if (classes[i].get\_meetingTime1().get\_id() not in classes[i].get\_room().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_AVAILABILITY, conflictBetweenClasses))  
 *# Limit Section Availibilty* if LIMIT\_SECTION\_AVAILABILITY == True:  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[i].get\_section().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_AVAILABILITY, conflictBetweenClasses))  
 else:  
 *# 0* if (classes[i].get\_meetingTime().get\_id() not in classes[i].get\_section().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_AVAILABILITY, conflictBetweenClasses))  
 *# 1* if (classes[i].get\_meetingTime1().get\_id() not in classes[i].get\_section().get\_availability()):  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_AVAILABILITY, conflictBetweenClasses))  
  
 for j in range(0, len(classes)):  
 *# Class Optimization* if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[j].get\_meetingTime().get\_restTime()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 if (classes[i].get\_meetingTime().get\_time() == classes[j].get\_meetingTime().get\_time()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 if (classes[i].get\_meetingTime().get\_time() in classes[j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 else:  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[j].get\_meetingTime().get\_restTime()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 if (classes[i].get\_meetingTime().get\_time() == classes[j].get\_meetingTime().get\_time()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 if (classes[i].get\_meetingTime().get\_time() in classes[j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 *# 11* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime1().get\_day() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[j].get\_meetingTime1().get\_restTime()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[j].get\_meetingTime1().get\_time()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[j].get\_meetingTime1().get\_breakTime()):  
 if ((classes[i].get\_meetingTime1().get\_MThour() + classes[j].get\_meetingTime1().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 *# 10* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime().get\_day() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[j].get\_meetingTime().get\_restTime()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[j].get\_meetingTime().get\_time()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime1().get\_MThour() + classes[j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 *# 01* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime1().get\_day() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[j].get\_meetingTime1().get\_restTime()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 if (classes[i].get\_meetingTime().get\_time() == classes[j].get\_meetingTime1().get\_time()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 if (classes[i].get\_meetingTime().get\_time() in classes[j].get\_meetingTime1().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[j].get\_meetingTime1().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.SECTION\_BOOKING,  
 conflictBetweenClasses))  
 *# Instructor Optimization* if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[j].get\_meetingTime().get\_restTime()):  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 if (classes[i].get\_meetingTime().get\_time() == classes[j].get\_meetingTime().get\_time()):  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 if (classes[i].get\_meetingTime().get\_time() in classes[j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 else:  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[j].get\_meetingTime().get\_restTime()):  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 if (classes[i].get\_meetingTime().get\_time() == classes[j].get\_meetingTime().get\_time()):  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 if (classes[i].get\_meetingTime().get\_time() in classes[j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 *# 11* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime1().get\_day() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[j].get\_meetingTime1().get\_restTime()):  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[j].get\_meetingTime1().get\_time()):  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[j].get\_meetingTime1().get\_breakTime()):  
 if ((classes[i].get\_meetingTime1().get\_MThour() + classes[j].get\_meetingTime1().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 *# 01* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime1().get\_day() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[j].get\_meetingTime1().get\_restTime()):  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 if (classes[i].get\_meetingTime().get\_time() == classes[j].get\_meetingTime1().get\_time()):  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 if (classes[i].get\_meetingTime().get\_time() in classes[j].get\_meetingTime1().get\_breakTime()):  
 if ((classes[i].get\_meetingTime().get\_MThour() + classes[j].get\_meetingTime1().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 *# 10* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime().get\_day() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[j].get\_meetingTime().get\_restTime()):  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[j].get\_meetingTime().get\_time()):  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 if (classes[i].get\_meetingTime1().get\_time() in classes[j].get\_meetingTime().get\_breakTime()):  
 if ((classes[i].get\_meetingTime1().get\_MThour() + classes[j].get\_meetingTime().get\_MThour()) > CONSECUTIVE\_TEACHING\_HOURS):  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 *# Room Optimization* if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[j].get\_meetingTime().get\_restTime()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 if (classes[i].get\_meetingTime().get\_time() == classes[j].get\_meetingTime().get\_time()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 else:  
 *# 00* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[j].get\_meetingTime().get\_restTime()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 if (classes[i].get\_meetingTime().get\_time() == classes[j].get\_meetingTime().get\_time()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 *# 11* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime1().get\_day() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[j].get\_meetingTime1().get\_restTime()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[j].get\_meetingTime1().get\_time()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 *# 01* if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime1().get\_day() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime().get\_time() in classes[j].get\_meetingTime1().get\_restTime()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 if (classes[i].get\_meetingTime().get\_time() == classes[j].get\_meetingTime1().get\_time()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 *# 10* if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime().get\_day() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room().get\_number() == classes[j].get\_room().get\_number()):  
 if (classes[i].get\_meetingTime1().get\_time() in classes[j].get\_meetingTime().get\_restTime()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 if (classes[i].get\_meetingTime1().get\_time() == classes[j].get\_meetingTime().get\_time()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 if (j >= i):  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_meetingTime().get\_id() == classes[j].get\_meetingTime().get\_id() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 else:  
 *# 00* if (classes[i].get\_meetingTime().get\_id() == classes[j].get\_meetingTime().get\_id() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 *# 11* if (classes[i].get\_meetingTime1().get\_id() == classes[j].get\_meetingTime1().get\_id() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 *# 01* if (classes[i].get\_meetingTime().get\_id() == classes[j].get\_meetingTime1().get\_id() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 *# 10* if (classes[i].get\_meetingTime1().get\_id() == classes[j].get\_meetingTime().get\_id() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_room() == classes[j].get\_room()):  
 if classes[i].get\_instructor().get\_seniority() < classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() > classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 elif classes[i].get\_instructor().get\_seniority() == classes[j].get\_instructor().get\_seniority():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.ROOM\_BOOKING, conflictBetweenClasses))  
 if (classes[i].get\_instructor() == classes[j].get\_instructor()):  
 if classes[i].get\_room().get\_type() == classes[j].get\_room().get\_type():  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_BOOKING,  
 conflictBetweenClasses))  
 *# Disable Dual Class Type* if DISABLE\_MIXED\_TYPE == True:  
 if (classes[i].get\_meetingTime1().get\_id() == **'NULL'**):  
 *# 00* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day()):  
 if (classes[i].get\_room().get\_type() != classes[j].get\_room().get\_type()):  
 if classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 conflictBetweenClasses)) *# Room Type Conflict* elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 conflictBetweenClasses)) *# Room Type Conflict* else:  
 *# 00* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime().get\_day()):  
 if (classes[i].get\_room().get\_type() != classes[j].get\_room().get\_type()):  
 if classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 conflictBetweenClasses)) *# Room Type Conflict* elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 conflictBetweenClasses)) *# Room Type Conflict  
 # 11* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime1().get\_day()):  
 if (classes[i].get\_room().get\_type() != classes[j].get\_room().get\_type()):  
 if classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 conflictBetweenClasses)) *# Room Type Conflict* elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 conflictBetweenClasses)) *# Room Type Conflict  
 # 01* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime().get\_day() == classes[j].get\_meetingTime1().get\_day()):  
 if (classes[i].get\_room().get\_type() != classes[j].get\_room().get\_type()):  
 if classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 conflictBetweenClasses)) *# Room Type Conflict* elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 conflictBetweenClasses)) *# Room Type Conflict  
 # 10* if (classes[i].get\_section().get\_code() == classes[j].get\_section().get\_code() and classes[i].get\_id() != classes[j].get\_id()):  
 if (classes[i].get\_meetingTime1().get\_day() == classes[j].get\_meetingTime().get\_day()):  
 if (classes[i].get\_room().get\_type() != classes[j].get\_room().get\_type()):  
 if classes[i].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 conflictBetweenClasses)) *# Room Type Conflict* elif classes[j].get\_room().get\_type() != **'PR'**:  
 conflictBetweenClasses=list()  
 conflictBetweenClasses.append(classes[j])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.MIXED\_TYPE,  
 conflictBetweenClasses)) *# Room Type Conflict* if (classes[j].get\_instructor().get\_name() == instructorNames[i]):  
 subjectHours.append(float(classes[j].get\_subject().get\_numbHour()))  
 *# Enable Unit Limit* instructor\_Hours.append(subjectHours)  
 if DISABLE\_INSTRUCTOR\_OVERLOAD == True:  
 if (classes[i].get\_instructor().get\_max\_hours() < sum(instructor\_Hours[i])):  
 instructorOverloadConflict=list()  
 instructorOverloadConflict.append(classes[i])  
 self.\_conflicts.append(  
 Conflict(Conflict.ConflictType.INSTRUCTOR\_OVERLOAD, instructorOverloadConflict))  
 return 1 / ((1.0 \* len(self.\_conflicts) + 1))  
  
 def \_\_str\_\_(self):  
 returnValue=**""** for i in range(0, len(self.\_classes) - 1):  
 returnValue+=str(self.\_classes[i]) + **", "** returnValue+=str(self.\_classes[len(self.\_classes) - 1])  
 return returnValue  
  
 class Population:  
 def \_\_init\_\_(self, size):  
 self.\_size=size  
 self.\_data=dbMgr  
 self.\_schedules=[]  
 for i in range(0, size): self.\_schedules.append(Schedule().initialize())  
  
 def get\_size(self): return self.\_size  
  
 def get\_schedules(self): return self.\_schedules  
  
 class GeneticAlgorithm:  
 def evolve(self, population):  
 return self.\_mutate\_population(self.\_crossover\_population(population))  
  
 def \_crossover\_population(self, pop):  
 crossover\_pop=Population(0)  
 for i in range(NUMB\_OF\_ELITE\_SCHEDULES):  
 crossover\_pop.get\_schedules().append(pop.get\_schedules()[i])  
 i=NUMB\_OF\_ELITE\_SCHEDULES  
 while i < POPULATION\_SIZE:  
 schedule1=self.\_select\_tournament\_population(pop).get\_schedules()[0]  
 schedule2=self.\_select\_tournament\_population(pop).get\_schedules()[0]  
 crossover\_pop.get\_schedules().append(self.\_crossover\_schedule(schedule1, schedule2))  
 i+=1  
 return crossover\_pop  
  
 def \_mutate\_population(self, population):  
 for i in range(NUMB\_OF\_ELITE\_SCHEDULES, POPULATION\_SIZE):  
 self.\_mutate\_schedule(population.get\_schedules()[i])  
 return population  
  
 def \_crossover\_schedule(self, schedule1, schedule2):  
 crossoverSchedule=Schedule().initialize()  
 for i in range(0, len(crossoverSchedule.get\_classes())):  
 if (rnd.random() > 0.5):  
 crossoverSchedule.get\_classes()[i]=schedule1.get\_classes()[i]  
 else:  
 crossoverSchedule.get\_classes()[i]=schedule2.get\_classes()[i]  
 return crossoverSchedule  
  
 def \_mutate\_schedule(self, mutateSchedule):  
 schedule=Schedule().initialize()  
 for i in range(0, len(mutateSchedule.get\_classes())):  
 if (MUTATION\_RATE > rnd.random()): mutateSchedule.get\_classes()[i]=schedule.get\_classes()[i]  
 return mutateSchedule  
  
 def \_select\_tournament\_population(self, pop):  
 tournament\_pop=Population(0)  
 i=0  
 while i < TOURNAMENT\_SELECTION\_SIZE:  
 tournament\_pop.get\_schedules().append(pop.get\_schedules()[rnd.randrange(0, int(POPULATION\_SIZE))])  
 i+=1  
 tournament\_pop.get\_schedules().sort(key = lambda x: x.get\_fitness(), reverse = True)  
 return tournament\_pop  
  
 class Section:  
 def \_\_init\_\_(self, code, subjects, availability):  
 self.\_code=code  
 self.\_subjects=subjects  
 self.\_availability=availability  
  
 def get\_code(self): return self.\_code  
  
 def get\_subjects(self): return self.\_subjects  
  
 def get\_availability(self): return self.\_availability  
  
 class Subject:  
 def \_\_init\_\_(self, code, description, units, numbHour, compatibility, curriculum, maxNumbOfStudents, dept,  
 instructors):  
 self.\_code=code  
 self.\_description=description  
 self.\_units=units  
 self.\_numbHour=numbHour  
 self.\_compatibility=compatibility  
 self.\_curriculum=curriculum  
 self.\_maxNumbOfStudents=maxNumbOfStudents  
 self.\_dept=dept  
 self.\_instructors=instructors  
  
 def get\_code(self): return self.\_code  
  
 def get\_description(self): return self.\_description  
  
 def get\_units(self): return self.\_units  
  
 def get\_numbHour(self): return self.\_numbHour  
  
 def get\_compatibility(self): return self.\_compatibility  
  
 def get\_curriculum(self): return self.\_curriculum  
  
 def get\_maxNumbOfStudents(self): return self.\_maxNumbOfStudents  
  
 def get\_dept(self): return self.\_dept  
  
 def get\_instructors(self): return self.\_instructors  
  
 def \_\_str\_\_(self): return self.\_description  
  
 class Time:  
 def \_\_init\_\_(self, block):  
 self.\_block=block  
  
 def get\_block(self): return self.\_block  
  
 def \_\_str\_\_(self): return self.\_block  
  
 class TimePoint:  
 def \_\_init\_\_(self, point, blocks, phantomBlocks):  
 self.\_point=point  
 self.\_blocks=blocks  
 self.\_phantomBlocks=phantomBlocks  
  
 def get\_point(self): return self.\_point  
  
 def get\_blocks(self): return self.\_blocks  
  
 def get\_phantomBlocks(self): return self.\_phantomBlocks  
  
 def \_\_str\_\_(self): return self.\_point  
  
 class Instructor:  
 def \_\_init\_\_(self, number, name, max\_hours, seniority, availability, start, end, overtime):  
 self.\_number=number  
 self.\_name=name  
 self.\_max\_hours=max\_hours  
 self.\_seniority=seniority  
 self.\_availability=availability  
 self.\_start=start  
 self.\_end=end  
 self.\_overtime=overtime  
  
 def get\_number(self): return self.\_number  
  
 def get\_name(self): return self.\_name  
  
 def get\_max\_hours(self): return self.\_max\_hours  
  
 def get\_seniority(self): return self.\_seniority  
  
 def get\_availability(self): return self.\_availability  
  
 def get\_start(self): return self.\_start  
  
 def get\_end(self): return self.\_end  
  
 def get\_overtime(self): return self.\_overtime  
  
 def \_\_str\_\_(self): return self.\_name  
  
 class MeetingTime:  
 def \_\_init\_\_(self, id, time, day, MThour, cell, restTime, breakTime):  
 self.\_id=id  
 self.\_time=time  
 self.\_day=day  
 self.\_MThour=MThour  
 self.\_cell=cell  
 self.\_restTime=restTime   
 self.\_breakTime=breakTime  
  
 def get\_id(self): return self.\_id  
  
 def get\_time(self): return self.\_time  
  
 def get\_day(self): return self.\_day  
  
 def get\_MThour(self): return self.\_MThour  
  
 def get\_cell(self): return self.\_cell  
  
 def get\_restTime(self): return self.\_restTime   
  
 def get\_breakTime(self): return self.\_breakTime  
  
 def \_\_str\_\_(self): return self.\_id  
  
 class MeetingTime1:  
 def \_\_init\_\_(self, id, time, day, MThour, cell, restTime, breakTime):  
 self.\_id=id  
 self.\_time=time  
 self.\_day=day  
 self.\_MThour=MThour  
 self.\_cell=cell  
 self.\_restTime=restTime   
 self.\_breakTime=breakTime  
  
 def get\_id(self): return self.\_id  
  
 def get\_time(self): return self.\_time  
  
 def get\_day(self): return self.\_day  
  
 def get\_MThour(self): return self.\_MThour  
  
 def get\_cell(self): return self.\_cell  
  
 def get\_restTime(self): return self.\_restTime   
  
 def get\_breakTime(self): return self.\_breakTime  
  
 def \_\_str\_\_(self): return self.\_id  
  
 class Room:  
 def \_\_init\_\_(self, number, seatingCapacity, type, dept, availability):  
 self.\_number=number  
 self.\_seatingCapacity=seatingCapacity  
 self.\_type=type  
 self.\_dept=dept  
 self.\_availability=availability  
  
 def get\_number(self): return self.\_number  
  
 def get\_seatingCapacity(self): return self.\_seatingCapacity  
  
 def get\_type(self): return self.\_type  
  
 def get\_dept(self): return self.\_dept  
  
 def get\_availability(self): return self.\_availability  
  
 def \_\_str\_\_(self): return self.\_number  
  
 class Class:  
 def \_\_init\_\_(self, id, section, subject):  
 self.\_id=id  
 self.\_section=section  
 self.\_subject=subject  
 self.\_instructor=None  
 self.\_meetingTime=None  
 self.\_meetingTime1=None  
 self.\_room=None  
  
 def get\_id(self): return self.\_id  
  
 def get\_section(self): return self.\_section  
  
 def get\_subject(self): return self.\_subject  
  
 def get\_instructor(self): return self.\_instructor  
  
 def get\_meetingTime(self): return self.\_meetingTime  
  
 def get\_meetingTime1(self): return self.\_meetingTime1  
  
 def get\_room(self): return self.\_room  
  
 def set\_instructor(self, instructor): self.\_instructor=instructor  
  
 def set\_meetingTime(self, meetingTime): self.\_meetingTime=meetingTime  
  
 def set\_meetingTime1(self, meetingTime1): self.\_meetingTime1=meetingTime1  
  
 def set\_room(self, room): self.\_room=room  
  
 def \_\_str\_\_(self):  
 return str(self.\_section.get\_code()) + **","** + str(self.\_subject.get\_code()) + **","** + \  
 str(self.\_room.get\_number()) + **","** + str(self.\_instructor.get\_number()) + **","** + str(  
 self.\_meetingTime.get\_id()) + **","** + str(self.\_meetingTime1.get\_id())  
  
 class Conflict:  
 class ConflictType(Enum):  
 WRONG\_MTHOUR=1  
 UNEQUAL\_SPLIT=2  
 CASUAL\_SPLITTING=3  
 SAME\_MTS=4  
 OVERLAP\_MTS=5  
  
 NUMB\_OF\_STUDENTS=6  
 ROOM\_AVAILABILITY=7  
 ROOM\_BOOKING=8  
 ROOM\_BOOKING\_OVERLAP=9  
 XDEPT\_ROOM\_UTILITY=10  
  
 INSTRUCTOR\_OVERTIME=11  
 INSTRUCTOR\_AVAILABILITY=12  
 INSTRUCTOR\_BOOKING=13  
 INSTRUCTOR\_OVERLOAD=14  
  
 SECTION\_AVAILABILITY=15  
 SECTION\_BOOKING=16  
  
 LAB\_ON\_VR=17  
 MIXED\_TYPE=18  
  
 def \_\_init\_\_(self, conflictType, conflictBetweenClasses):  
 self.\_conflictType=conflictType  
 self.\_conflictBetweenClasses=conflictBetweenClasses  
  
 def get\_conflictType(self): return self.\_conflictType  
  
 def get\_conflictBetweenClasses(self): return self.\_conflictBetweenClasses  
  
 def \_\_str\_\_(self): return str(self.\_conflictType) + **" "** + str(  
 **" and "**.join(map(str, self.\_conflictBetweenClasses)))  
  
 class DisplayMgr:  
 @staticmethod  
 def display\_input\_data():  
 print(**"> All Available Data"**)  
 DisplayMgr.display\_section()  
 DisplayMgr.display\_room()  
 DisplayMgr.display\_instructor()  
 DisplayMgr.display\_subject()  
  
 @staticmethod  
 def display\_section():  
 sections=dbMgr.get\_sections()  
 availableSectionsTable=prettytable.PrettyTable([**'section'**, **'subjects'**])  
 for i in range(0, len(sections)):  
 subjects=sections.\_\_getitem\_\_(i).get\_subjects()  
 tempStr=**"["** for j in range(0, len(subjects) - 1):  
 tempStr+=subjects[j].\_\_str\_\_() + **", "** tempStr+=subjects[len(subjects) - 1].\_\_str\_\_() + **"]"** availableSectionsTable.add\_row([sections.\_\_getitem\_\_(i).get\_code(), tempStr])  
 print(availableSectionsTable)  
  
 @staticmethod  
 def display\_subject():  
 availableSubjectsTable=prettytable.PrettyTable(  
 [**'ID'**, **'Subject'**, **'Units'**, **'Hours'**, **'Compatibility'**, **'Curriculum'**, **'Max. # of Students'**, **'Dept'**])  
 subjects=dbMgr.get\_subjects()  
 for i in range(0, len(subjects)):  
 availableSubjectsTable.add\_row(  
 [subjects[i].get\_code(), subjects[i].get\_description(), subjects[i].get\_units(),  
 subjects[i].get\_numbHour(),  
 subjects[i].get\_compatibility(), subjects[i].get\_curriculum(),  
 str(subjects[i].get\_maxNumbOfStudents()), str(subjects[i].get\_dept())])   
 print(availableSubjectsTable)  
  
 @staticmethod  
 def display\_instructor():  
 availableInstructorsTable=prettytable.PrettyTable([**'ID'**, **'Instructor'**, **'Max Hours'**, **'Seniority'**])  
 instructors=dbMgr.get\_instructors()  
 for i in range(0, len(instructors)):  
 availableInstructorsTable.add\_row(  
 [instructors[i].get\_number(), instructors[i].get\_name(), instructors[i].get\_max\_hours(),  
 instructors[i].get\_seniority()])  
  
 @staticmethod  
 def display\_room():  
 availableRoomsTable=prettytable.PrettyTable([**'Room'**, **'Max. Capacity'**, **'Type'**, **'Dept'**])  
 rooms=dbMgr.get\_rooms()  
 for i in range(0, len(rooms)):  
 availableRoomsTable.add\_row(  
 [str(rooms[i].get\_number()), str(rooms[i].get\_seatingCapacity()), str(rooms[i].get\_type()),  
 str(rooms[i].get\_dept())])  
 print(availableRoomsTable)  
  
 @staticmethod  
 def display\_meeting\_times():  
 availableMeetingTimeTable=prettytable.PrettyTable([**'ID'**, **'Meeting Time'**, **'Rest Time'**])  
 meetingTimes=dbMgr.get\_meetingTimes()  
 for i in range(0, len(meetingTimes)):  
 restTime=[]  
 for j in range(0, len(meetingTimes[i].get\_restTime())):  
 restTime.append(meetingTimes[i].get\_restTime()[j])  
 availableMeetingTimeTable.add\_row([meetingTimes[i].get\_id(), meetingTimes[i].get\_time(), restTime])  
 print(availableMeetingTimeTable)  
  
 @staticmethod  
 def display\_generation(population):  
 table1=prettytable.PrettyTable(  
 [**'schedule #'**, **'fitness'**, **'# of conflicts'**, **'classes [section,class,room,instructor,meeting-time]'**])  
 schedules=population.get\_schedules()  
 for i in range(0, len(schedules)):  
 table1.add\_row([str(i + 1), round(schedules[i].get\_fitness(), 3), len(schedules[i].get\_conflicts()),  
 schedules[i].\_\_str\_\_()])  
 print(table1)  
  
 @staticmethod  
 def display\_schedule\_as\_table(schedule):  
 classes=schedule.get\_classes()  
 table=prettytable.PrettyTable(  
 [**'Class #'**, **'Section'**, **'Subject (number, max # of students)'**, **'Room (Capacity)'**, **'Instructor (Id)'**,  
 **'Meeting Time 0 (ID)'**, **'Meeting Time 1 (ID)'**])  
 for i in range(0, len(classes)):  
 table.add\_row([str(i + 1), str(classes[i].get\_section().get\_code()),  
 str(classes[i].get\_subject().get\_description()) + **" ("** +  
 str(classes[i].get\_subject().get\_code()) + **", "** +  
 str(classes[i].get\_subject().get\_maxNumbOfStudents()) + **")"**,  
 str(classes[i].get\_room().get\_number()) + **" ("** + str(  
 classes[i].get\_room().get\_seatingCapacity()) + **")"**,  
 str(classes[i].get\_instructor().get\_name()) + **" ("** + str(  
 classes[i].get\_instructor().get\_number()) + **")"**,  
 str(classes[i].get\_meetingTime().get\_time()) + **" ("** + str(  
 classes[i].get\_meetingTime().get\_id()) + **")"**,  
 str(classes[i].get\_meetingTime1().get\_time()) + **" ("** + str(  
 classes[i].get\_meetingTime1().get\_id()) + **")"**])  
 print(table)  
  
 *# Perspectives* @staticmethod  
 def SECTIONXLSX(schedule):  
 sections=dbMgr.get\_sections()  
 sectionCodes=list()  
 section\_instructorBlocks=list()  
 MT\_timeBlocks=list()  
 section\_rooms=list()  
 section\_subjects=list()  
 for i in range(0, len(sections)):  
 sectionCodes.append(sections[i].get\_code())  
 instructorBlock=list()  
 timeBlock=list()  
 roomBlock=list()  
 subjectBlock=list()  
 for j in range(0, len(schedule.get\_classes())):  
 if (schedule.get\_classes()[j].get\_section() == sections[i]):  
 instructorBlock.append(str(schedule.get\_classes()[j].get\_instructor().get\_number()))  
 timeBlock.append(str(schedule.get\_classes()[j].get\_meetingTime().get\_id()))  
 roomBlock.append(str(schedule.get\_classes()[j].get\_room().get\_number()))  
 subjectBlock.append(str(schedule.get\_classes()[j].get\_subject().get\_code()))  
 for j in range(0, len(schedule.get\_classes())):  
 if (schedule.get\_classes()[j].get\_subject().get\_numbHour() > schedule.get\_classes()[  
 j].get\_meetingTime().get\_MThour()):  
 if (schedule.get\_classes()[j].get\_section() == sections[i]):  
 instructorBlock.append(str(schedule.get\_classes()[j].get\_instructor().get\_number()))  
 timeBlock.append(str(schedule.get\_classes()[j].get\_meetingTime1().get\_id()))  
 roomBlock.append(str(schedule.get\_classes()[j].get\_room().get\_number()))  
 subjectBlock.append(str(schedule.get\_classes()[j].get\_subject().get\_code()))  
 section\_instructorBlocks.append(instructorBlock)  
 MT\_timeBlocks.append(timeBlock)  
 section\_rooms.append(roomBlock)  
 section\_subjects.append(subjectBlock)  
 timeCells=dbMgr.get\_meetingTimes()  
 meeetingTimes=list()  
 cells=list()  
 for i in range(0, len(timeCells)):  
 meeetingTimes.append(timeCells[i].get\_id())  
 cells.append(timeCells[i].get\_cell())  
  
 *# create file (workbook) and worksheet* outWorkbook=xlsxwriter.Workbook(**"section.xlsx"**)  
  
 def SECTIONBLOCKS():  
 *# declare data* columnData=[**"Time"**, **"Monday"**, **"Tuesday"**, **"Wednesday"**, **"Thursday"**, **"Friday"**, **"Saturday"**, **"Sunday"**]  
 timeBlock=[**"7:00 - 7:30"**, **"7:30 - 8:00"**, **"8:00 - 8:30"**, **"8:30 - 9:00"**, **"9:00 - 9:30"**,  
 **"9:30 - 10:00"**,  
 **"10:00 - 10:30"**,  
 **"10:30 - 11:00"**, **"11:00 - 11:30"**, **"11:30 - 12:00"**, **"12:00 - 12:30"**, **"12:30 - 1:00"**,  
 **"1:00 - 1:30"**,  
 **"1:30 -2:00"**, **"2:00 - 2:30"**, **"2:30 - 3:00"**, **"3:00 - 3:30"**, **"3:30 - 4:00"**,  
 **"4:00 - 4:30"**,  
 **"4:30 - 5:00"**,  
 **"5:00 - 5:30"**, **"5:30 - 6:00"**, **"6:00 - 6:30"**, **"6:30 - 7:00"**, **"7:00 - 7:30"**,  
 **"7:30 - 8:00"**,  
 **"8:00 - 8:30"**,  
 **"8:30 - 9:00"**]  
 load\_title=**'STUDENT SCHEDULE'** summaryColumn=[**"SUBJECT"**, **"TIME"**, **"DAY"**, **"ROOM"**, **"UNIT"**, **"# OF STUDENTS"**, **"FACULTY"**]  
 outputSheet=outWorkbook.add\_worksheet(name = sectionCodes[sheet])  
 *# Increase the cell size of the merged cells to highlight the formatting.* outputSheet.set\_column(**'A:H'**, 15)  
 outputSheet.set\_default\_row(20)  
 *# Create a format to use in the...  
 # ...centerBoldHead.* centerBoldHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...centerBoldHeadless.* centerBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...leftBoldHeadless.* leftBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...uleftHeadless.* uleftHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...rightBoldHeadless.* rightBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 0,  
 **'align'**: **'right'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...centerHead.* centerHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...ucenterHeadless.* ucenterHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...icenterHeadless.* icenterHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'italic'**: 1,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...smallcenterHead.* smallcenterHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 7.5,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...xsmallcenterHead.* xsmallcenterHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 7.5,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'fadfca'**})  
 *# ...ysmallcenterHead.* ysmallcenterHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 7.5,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'ffe338'**})  
 *# ...leftHeadless.* leftHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...details.* sheet\_details=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vbottom'**,  
 **'fg\_color'**: **'white'**})  
 *# ...header.* header=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 22,  
 **'font\_color'**: **'white'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'c31d3a'**}) *# Green, Accent 6, Darker 25%: 76933C  
 # ... columns.* column=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'white'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'c41e39'**}) *# Green, Accent 6: 9BBB59  
 # ...time.* time=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**}) *# Green, Accent 6, Lighter 40%: C4D798  
 # ...classSched.* classSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'ececec'**}) *# Emerald: 50C878  
 # Write inside the Cell  
 # ...columnData* for item in range(len(columnData)):  
 outputSheet.write(11, item + 0, columnData[item], column)  
 *# ...timeBlock* for item in range(len(timeBlock)):  
 outputSheet.write(item + 12, 0, timeBlock[item], time)  
 *# ...Merge School* outputSheet.merge\_range(**'B1:F4'**, **''**, centerHead)  
 outputSheet.write\_rich\_string(**'B1'**, centerBoldHead, schooltext, centerHead, school\_infotext,  
 centerHead)  
 *# ...Load Title* outputSheet.merge\_range(**'B5:F6'**, load\_title, centerBoldHead)  
 *# ...VAA-OAP* outputSheet.merge\_range(**'A5:A6'**, vaa\_oap, centerHead)  
 *# ...Sheet Details* outputSheet.write\_string(**'G1'**, **'Index No.'**, sheet\_details)  
 outputSheet.write\_string(**'G2'**, **'Issue No.'**, sheet\_details)  
 outputSheet.write\_string(**'G3'**, **'Revision No.'**, sheet\_details)  
 outputSheet.write\_string(**'G4'**, **'Date'**, sheet\_details)  
 outputSheet.write\_string(**'G5'**, **'Page'**, sheet\_details)  
 outputSheet.write\_string(**'G6'**, **'QAC No.'**, sheet\_details)  
 outputSheet.write\_string(**'H1'**, str(indexno), sheet\_details)  
 outputSheet.write\_string(**'H2'**, str(issueno), sheet\_details)  
 outputSheet.write\_string(**'H3'**, str(revisionno), sheet\_details)  
 outputSheet.write\_string(**'H4'**, str(date), sheet\_details)  
 outputSheet.write\_string(**'H5'**, str(page), sheet\_details)  
 outputSheet.write\_string(**'H6'**, str(qacno), sheet\_details)  
 *# ...Logo* outputSheet.merge\_range(**'A1:A4'**, **''**, centerHead)  
 outputSheet.insert\_image(**'A1'**, **'tup-logo.png'**, {**'x\_scale'**: 1.1, **'y\_scale'**: 1.1})  
 *# ...College* outputSheet.merge\_range(**'A8:B8'**, **' COLLEGE:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C8:E8'**,  
 **'\_'** + college + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...Department* outputSheet.merge\_range(**'A9:B9'**, **' DEPARTMENT:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C9:E9'**,  
 **'\_'** + department + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...Name of Faculty* outputSheet.merge\_range(**'A10:B10'**, **' SECTION:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C10:E10'**,  
 **'\_'** + str(  
 sectionCodes[sheet]) + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...S.Y.* outputSheet.write\_string(**'G8'**, **'S.Y.:'**, rightBoldHeadless)  
 outputSheet.write\_string(**'H8'**, **'\_\_'** + schoolyear + **'\_\_'**, uleftHeadless)  
 *# ...Semester* outputSheet.write\_string(**'G9'**, **'Semester:'**, rightBoldHeadless)  
 outputSheet.write\_string(**'H9'**, **'\_'** + semester + **'\_'**, uleftHeadless)  
 *# ...bordercells* for row in range(12, 40):  
 for col in range(1, 8):  
 outputSheet.write(row, col, **''**, centerHead)  
 *# Summary Table  
 # ...summaryColumn* for item in range(len(summaryColumn)):  
 outputSheet.write(41, item, summaryColumn[item], column)  
 outputSheet.merge\_range(41, 6, 41, 7, summaryColumn[6], column)  
  
 for row in range(42, 54):  
 for col in range(0, 6):  
 outputSheet.write(row, col, **''**, centerHead)  
 outputSheet.merge\_range(row, 6, row, 7, **''**, centerHead)  
 classes=schedule.get\_classes()  
 class\_list=list()  
 for i in range(0, len(classes)):  
 if (classes[i].get\_section().get\_code() == sectionCodes[sheet]):  
 class\_list.append(classes[i])  
 for i in range(0, len(class\_list)):  
 if class\_list[i].get\_section().get\_code() == sectionCodes[sheet]:  
 outputSheet.write(42 + i, 0, class\_list[i].get\_subject().get\_code(), smallcenterHead)  
 if class\_list[i].get\_meetingTime1().get\_id() != **'NULL'**:  
 outputSheet.write(42 + i, 1,  
 str(class\_list[i].get\_meetingTime().get\_time()) + **'**\n**'** + str(  
 class\_list[i].get\_meetingTime1().get\_time()), smallcenterHead)  
 outputSheet.write(42 + i, 2,  
 str(class\_list[i].get\_meetingTime().get\_day()[0:3]) + **'**\n**'** + str(  
 class\_list[i].get\_meetingTime1().get\_day()[0:3]), smallcenterHead)  
 else:  
 outputSheet.write(42 + i, 1, class\_list[i].get\_meetingTime().get\_time(),  
 smallcenterHead)  
 outputSheet.write(42 + i, 2, class\_list[i].get\_meetingTime().get\_day()[0:3],  
 smallcenterHead)  
 outputSheet.write(42 + i, 3, class\_list[i].get\_room().get\_number(), smallcenterHead)  
 outputSheet.write(42 + i, 4, class\_list[i].get\_subject().get\_units(), smallcenterHead)  
 outputSheet.write(42 + i, 5, class\_list[i].get\_subject().get\_maxNumbOfStudents(),  
 smallcenterHead)  
 outputSheet.write(42 + i, 6, class\_list[i].get\_instructor().get\_name(), smallcenterHead)  
  
 *# ...Authority Sign* outputSheet.merge\_range(**'A55:C55'**, **' Prepared By:'**, leftHeadless)  
 outputSheet.merge\_range(**'A57:C57'**, **'\_\_\_\_\_'** + str(depthead) + **'\_\_\_\_\_'**, ucenterHeadless)  
 outputSheet.merge\_range(**'A58:C58'**, **'Department Head'**, icenterHeadless)  
 outputSheet.merge\_range(**'F55:H55'**, **' Recommending Approval:'**, leftHeadless)  
 outputSheet.merge\_range(**'F57:H57'**, **'\_\_\_\_\_'** + str(dean) + **'\_\_\_\_\_'**, ucenterHeadless)  
 outputSheet.merge\_range(**'F58:H58'**, **'College Dean'**, icenterHeadless)  
 *# ...footer* outputSheet.write\_string(**'B60'**, **'Transaction ID'**, sheet\_details)  
 outputSheet.merge\_range(**'C60:G60'**, **''**, sheet\_details)  
 outputSheet.write\_string(**'B61'**, **'Signature'**, sheet\_details)  
 outputSheet.merge\_range(**'C61:G61'**, **''**, sheet\_details)  
  
 for MT in MT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in MT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 section\_instructorBlocks[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 section\_rooms[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 section\_subjects[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 MT\_timeBlocks[sheet][MT\_timeBlocks[sheet].index(MT)]), classSched)  
  
 for numb\_of\_sections in range(0, len(sections)):  
 sheet=numb\_of\_sections  
 SECTIONBLOCKS()  
 outWorkbook.close()  
  
 @staticmethod  
 def ROOMXLSX(schedule):  
 rooms=dbMgr.get\_rooms()  
 roomNumbers=list()  
 room\_sectionBlocks=list()  
 MT\_timeBlocks=list()  
 room\_instructors=list()  
 room\_subjects=list()  
 for i in range(0, len(rooms)):  
 roomNumbers.append(rooms[i].get\_number())  
 sectionBlock=list()  
 instructorBlock=list()  
 subjectBlock=list()  
 timeBlock=list()  
 for j in range(0, len(schedule.get\_classes())):  
 if (schedule.get\_classes()[j].get\_room() == rooms[i]):  
 sectionBlock.append(str(schedule.get\_classes()[j].get\_section().get\_code()))  
 instructorBlock.append(str(schedule.get\_classes()[j].get\_instructor().get\_number()))  
 subjectBlock.append(str(schedule.get\_classes()[j].get\_subject().get\_code()))  
 timeBlock.append(str(schedule.get\_classes()[j].get\_meetingTime().get\_id()))  
 for j in range(0, len(schedule.get\_classes())):  
 if (schedule.get\_classes()[j].get\_subject().get\_numbHour() > schedule.get\_classes()[  
 j].get\_meetingTime().get\_MThour()):  
 if (schedule.get\_classes()[j].get\_room() == rooms[i]):  
 sectionBlock.append(str(schedule.get\_classes()[j].get\_section().get\_code()))  
 instructorBlock.append(str(schedule.get\_classes()[j].get\_instructor().get\_number()))  
 subjectBlock.append(str(schedule.get\_classes()[j].get\_subject().get\_code()))  
 timeBlock.append(str(schedule.get\_classes()[j].get\_meetingTime1().get\_id()))  
 room\_sectionBlocks.append(sectionBlock)  
 room\_instructors.append(instructorBlock)  
 room\_subjects.append(subjectBlock)  
 MT\_timeBlocks.append(timeBlock)  
 timeCells=dbMgr.get\_meetingTimes()  
 meeetingTimes=list()  
 cells=list()  
 for i in range(0, len(timeCells)):  
 meeetingTimes.append(timeCells[i].get\_id())  
 cells.append(timeCells[i].get\_cell())  
  
 *# create file (workbook) and worksheet* outWorkbook=xlsxwriter.Workbook(**"room.xlsx"**)  
  
 def ROOMBLOCKS():  
 *# declare data* columnData=[**"Time"**, **"Monday"**, **"Tuesday"**, **"Wednesday"**, **"Thursday"**, **"Friday"**, **"Saturday"**, **"Sunday"**]  
 timeBlock=[**"7:00 - 7:30"**, **"7:30 - 8:00"**, **"8:00 - 8:30"**, **"8:30 - 9:00"**, **"9:00 - 9:30"**,  
 **"9:30 - 10:00"**,  
 **"10:00 - 10:30"**,  
 **"10:30 - 11:00"**, **"11:00 - 11:30"**, **"11:30 - 12:00"**, **"12:00 - 12:30"**, **"12:30 - 1:00"**,  
 **"1:00 - 1:30"**,  
 **"1:30 -2:00"**, **"2:00 - 2:30"**, **"2:30 - 3:00"**, **"3:00 - 3:30"**, **"3:30 - 4:00"**,  
 **"4:00 - 4:30"**,  
 **"4:30 - 5:00"**,  
 **"5:00 - 5:30"**, **"5:30 - 6:00"**, **"6:00 - 6:30"**, **"6:30 - 7:00"**, **"7:00 - 7:30"**,  
 **"7:30 - 8:00"**,  
 **"8:00 - 8:30"**,  
 **"8:30 - 9:00"**]  
 load\_title=**'ROOM SCHEDULE'** outputSheet=outWorkbook.add\_worksheet(name = roomNumbers[sheet])  
 *# Increase the cell size of the merged cells to highlight the formatting.* outputSheet.set\_column(**'A:H'**, 15)  
 outputSheet.set\_default\_row(20)  
 *# Create a format to use in the...  
 # ...centerBoldHead.* centerBoldHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...centerBoldHeadless.* centerBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...leftBoldHeadless.* leftBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...uleftHeadless.* uleftHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...rightBoldHeadless.* rightBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 0,  
 **'align'**: **'right'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...centerHead.* centerHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...ucenterHeadless.* ucenterHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...icenterHeadless.* icenterHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'italic'**: 1,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...smallcenterHead.* smallcenterHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 7.5,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...leftHeadless.* leftHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...details.* sheet\_details=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vbottom'**,  
 **'fg\_color'**: **'white'**})  
 *# ...header.* header=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 22,  
 **'font\_color'**: **'white'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'c31d3a'**}) *# Green, Accent 6, Darker 25%: 76933C  
 # ... columns.* column=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'white'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'c41e39'**}) *# Green, Accent 6: 9BBB59  
 # ...time.* time=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**}) *# Green, Accent 6, Lighter 40%: C4D798  
 # ...classSched.* classSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'ececec'**}) *# Emerald: 50C878  
 # Write inside the Cell  
 # ...columnData* for item in range(len(columnData)):  
 outputSheet.write(11, item + 0, columnData[item], column)  
 *# ...timeBlock* for item in range(len(timeBlock)):  
 outputSheet.write(item + 12, 0, timeBlock[item], time)  
 *# ...Merge School* outputSheet.merge\_range(**'B1:F4'**, **''**, centerHead)  
 outputSheet.write\_rich\_string(**'B1'**, centerBoldHead, schooltext, centerHead, school\_infotext,  
 centerHead)  
 *# ...Load Title* outputSheet.merge\_range(**'B5:F6'**, load\_title, centerBoldHead)  
 *# ...VAA-OAP* outputSheet.merge\_range(**'A5:A6'**, vaa\_oap, centerHead)  
 *# ...Sheet Details* outputSheet.write\_string(**'G1'**, **'Index No.'**, sheet\_details)  
 outputSheet.write\_string(**'G2'**, **'Issue No.'**, sheet\_details)  
 outputSheet.write\_string(**'G3'**, **'Revision No.'**, sheet\_details)  
 outputSheet.write\_string(**'G4'**, **'Date'**, sheet\_details)  
 outputSheet.write\_string(**'G5'**, **'Page'**, sheet\_details)  
 outputSheet.write\_string(**'G6'**, **'QAC No.'**, sheet\_details)  
 outputSheet.write\_string(**'H1'**, str(indexno), sheet\_details)  
 outputSheet.write\_string(**'H2'**, str(issueno), sheet\_details)  
 outputSheet.write\_string(**'H3'**, str(revisionno), sheet\_details)  
 outputSheet.write\_string(**'H4'**, str(date), sheet\_details)  
 outputSheet.write\_string(**'H5'**, str(page), sheet\_details)  
 outputSheet.write\_string(**'H6'**, str(qacno), sheet\_details)  
 *# ...Logo* outputSheet.merge\_range(**'A1:A4'**, **''**, centerHead)  
 outputSheet.insert\_image(**'A1'**, **'tup-logo.png'**, {**'x\_scale'**: 1.1, **'y\_scale'**: 1.1})  
 *# ...College* outputSheet.merge\_range(**'A8:B8'**, **' COLLEGE:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C8:E8'**,  
 **'\_'** + college + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...Department* outputSheet.merge\_range(**'A9:B9'**, **' DEPARTMENT:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C9:E9'**,  
 **'\_'** + department + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...Name of Faculty* outputSheet.merge\_range(**'A10:B10'**, **' ROOM:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C10:E10'**,  
 **'\_'** + str(  
 roomNumbers[sheet]) + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...S.Y.* outputSheet.write\_string(**'G8'**, **'S.Y.:'**, rightBoldHeadless)  
 outputSheet.write\_string(**'H8'**, **'\_\_'** + schoolyear + **'\_\_'**, uleftHeadless)  
 *# ...Semester* outputSheet.write\_string(**'G9'**, **'Semester:'**, rightBoldHeadless)  
 outputSheet.write\_string(**'H9'**, **'\_'** + semester + **'\_'**, uleftHeadless)  
 *# ...bordercells* for row in range(12, 40):  
 for col in range(1, 8):  
 outputSheet.write(row, col, **''**, centerHead)  
 *# ...Authority Sign* outputSheet.merge\_range(**'A42:C42'**, **' Prepared By:'**, leftHeadless)  
 outputSheet.merge\_range(**'A44:C44'**, **'\_\_\_\_\_'** + str(depthead) + **'\_\_\_\_\_'**, ucenterHeadless)  
 outputSheet.merge\_range(**'A45:C45'**, **'Department Head'**, icenterHeadless)  
 outputSheet.merge\_range(**'F42:H42'**, **' Recommending Approval:'**, leftHeadless)  
 outputSheet.merge\_range(**'F44:H44'**, **'\_\_\_\_\_'** + str(dean) + **'\_\_\_\_\_'**, ucenterHeadless)  
 outputSheet.merge\_range(**'F45:H45'**, **'College Dean'**, icenterHeadless)  
 *# ...footer* outputSheet.write\_string(**'B48'**, **'Transaction ID'**, sheet\_details)  
 outputSheet.merge\_range(**'C48:G48'**, **''**, sheet\_details)  
 outputSheet.write\_string(**'B49'**, **'Signature'**, sheet\_details)  
 outputSheet.merge\_range(**'C49:G49'**, **''**, sheet\_details)  
  
 for MT in MT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in MT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 room\_instructors[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 room\_sectionBlocks[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 room\_subjects[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 MT\_timeBlocks[sheet][MT\_timeBlocks[sheet].index(MT)]), classSched)  
  
 for numb\_of\_rooms in range(0, len(rooms)):  
 sheet=numb\_of\_rooms  
 ROOMBLOCKS()  
 outWorkbook.close()  
  
 @staticmethod  
 def INSTRUCTORXLSX(schedule):  
 instructors=dbMgr.get\_instructors()  
 instructorIDs=list()  
 instructor\_sectionBlocks=list()  
 MT\_timeBlocks=list()  
 instructor\_rooms=list()  
 instructor\_subjects=list()  
 for i in range(0, len(instructors)):  
 instructorIDs.append(instructors[i].get\_name())  
 sectionBlock=list()  
 timeBlock=list()  
 roomBlock=list()  
 subjectBlock=list()  
 for j in range(0, len(schedule.get\_classes())):  
 if (schedule.get\_classes()[j].get\_instructor() == instructors[i]):  
 sectionBlock.append(str(schedule.get\_classes()[j].get\_section().get\_code()))  
 timeBlock.append(str(schedule.get\_classes()[j].get\_meetingTime().get\_id()))  
 roomBlock.append(str(schedule.get\_classes()[j].get\_room().get\_number()))  
 subjectBlock.append(str(schedule.get\_classes()[j].get\_subject().get\_code()))  
 for j in range(0, len(schedule.get\_classes())):  
 if (schedule.get\_classes()[j].get\_subject().get\_numbHour() > schedule.get\_classes()[  
 j].get\_meetingTime().get\_MThour()):  
 if (schedule.get\_classes()[j].get\_instructor() == instructors[i]):  
 sectionBlock.append(str(schedule.get\_classes()[j].get\_section().get\_code()))  
 timeBlock.append(str(schedule.get\_classes()[j].get\_meetingTime1().get\_id()))  
 roomBlock.append(str(schedule.get\_classes()[j].get\_room().get\_number()))  
 subjectBlock.append(str(schedule.get\_classes()[j].get\_subject().get\_code()))  
 instructor\_sectionBlocks.append(sectionBlock)  
 MT\_timeBlocks.append(timeBlock)  
 instructor\_rooms.append(roomBlock)  
 instructor\_subjects.append(subjectBlock)  
 timeCells=dbMgr.get\_meetingTimes()  
 meeetingTimes=list()  
 cells=list()  
 for i in range(0, len(timeCells)):  
 meeetingTimes.append(timeCells[i].get\_id())  
 cells.append(timeCells[i].get\_cell())  
  
 *# create file (workbook) and worksheet* outWorkbook=xlsxwriter.Workbook(**"instructor.xlsx"**)  
  
 def INSTRUCTORBLOCKS():  
 *# declare data* columnData=[**"Time"**, **"Monday"**, **"Tuesday"**, **"Wednesday"**, **"Thursday"**, **"Friday"**, **"Saturday"**, **"Sunday"**]  
 timeBlock=[**"7:00 - 7:30"**, **"7:30 - 8:00"**, **"8:00 - 8:30"**, **"8:30 - 9:00"**, **"9:00 - 9:30"**,  
 **"9:30 - 10:00"**,  
 **"10:00 - 10:30"**,  
 **"10:30 - 11:00"**, **"11:00 - 11:30"**, **"11:30 - 12:00"**, **"12:00 - 12:30"**, **"12:30 - 1:00"**,  
 **"1:00 - 1:30"**,  
 **"1:30 -2:00"**, **"2:00 - 2:30"**, **"2:30 - 3:00"**, **"3:00 - 3:30"**, **"3:30 - 4:00"**,  
 **"4:00 - 4:30"**,  
 **"4:30 - 5:00"**,  
 **"5:00 - 5:30"**, **"5:30 - 6:00"**, **"6:00 - 6:30"**, **"6:30 - 7:00"**, **"7:00 - 7:30"**,  
 **"7:30 - 8:00"**,  
 **"8:00 - 8:30"**,  
 **"8:30 - 9:00"**]  
 load\_title=**'INDIVIDUAL FACULTY LOADING'** outputSheet=outWorkbook.add\_worksheet(name = instructorIDs[sheet])  
 *# Increase the cell size of the merged cells to highlight the formatting.* outputSheet.set\_column(**'A:H'**, 15)  
 outputSheet.set\_default\_row(20)  
 *# Create a format to use in the...  
 # ...centerBoldHead.* centerBoldHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...centerBoldHeadless.* centerBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...leftBoldHeadless.* leftBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...uleftHeadless.* uleftHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...rightBoldHeadless.* rightBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 0,  
 **'align'**: **'right'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...centerHead.* centerHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...ucenterHeadless.* ucenterHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...icenterHeadless.* icenterHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'italic'**: 1,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...smallcenterHead.* smallcenterHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 7.5,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...leftHeadless.* leftHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...details.* sheet\_details=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vbottom'**,  
 **'fg\_color'**: **'white'**})  
 *# ...header.* header=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 22,  
 **'font\_color'**: **'white'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'c31d3a'**}) *# Green, Accent 6, Darker 25%: 76933C  
 # ... columns.* column=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'white'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'c41e39'**}) *# Green, Accent 6: 9BBB59  
 # ...time.* time=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**}) *# Green, Accent 6, Lighter 40%: C4D798  
 # ...classSched.* classSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'ececec'**}) *# Emerald: 50C878  
 # Write inside the Cell  
 # ...columnData* for item in range(len(columnData)):  
 outputSheet.write(11, item + 0, columnData[item], column)  
 *# ...timeBlock* for item in range(len(timeBlock)):  
 outputSheet.write(item + 12, 0, timeBlock[item], time)  
 *# ...Merge School* outputSheet.merge\_range(**'B1:F4'**, **''**, centerHead)  
 outputSheet.write\_rich\_string(**'B1'**, centerBoldHead, schooltext, centerHead, school\_infotext,  
 centerHead)  
 *# ...Load Title* outputSheet.merge\_range(**'B5:F6'**, load\_title, centerBoldHead)  
 *# ...VAA-OAP* outputSheet.merge\_range(**'A5:A6'**, vaa\_oap, centerHead)  
 *# ...Sheet Details* outputSheet.write\_string(**'G1'**, **'Index No.'**, sheet\_details)  
 outputSheet.write\_string(**'G2'**, **'Issue No.'**, sheet\_details)  
 outputSheet.write\_string(**'G3'**, **'Revision No.'**, sheet\_details)  
 outputSheet.write\_string(**'G4'**, **'Date'**, sheet\_details)  
 outputSheet.write\_string(**'G5'**, **'Page'**, sheet\_details)  
 outputSheet.write\_string(**'G6'**, **'QAC No.'**, sheet\_details)  
 outputSheet.write\_string(**'H1'**, str(indexno), sheet\_details)  
 outputSheet.write\_string(**'H2'**, str(issueno), sheet\_details)  
 outputSheet.write\_string(**'H3'**, str(revisionno), sheet\_details)  
 outputSheet.write\_string(**'H4'**, str(date), sheet\_details)  
 outputSheet.write\_string(**'H5'**, str(page), sheet\_details)  
 outputSheet.write\_string(**'H6'**, str(qacno), sheet\_details)  
 *# ...Logo* outputSheet.merge\_range(**'A1:A4'**, **''**, centerHead)  
 outputSheet.insert\_image(**'A1'**, **'tup-logo.png'**, {**'x\_scale'**: 1.1, **'y\_scale'**: 1.1})  
 *# ...College* outputSheet.merge\_range(**'A8:B8'**, **' COLLEGE:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C8:E8'**,  
 **'\_'** + college + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...Department* outputSheet.merge\_range(**'A9:B9'**, **' DEPARTMENT:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C9:E9'**,  
 **'\_'** + department + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...Name of Faculty* outputSheet.merge\_range(**'A10:B10'**, **' NAME OF FACULTY:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C10:E10'**,  
 **'\_'** + str(  
 instructorIDs[sheet]) + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...S.Y.* outputSheet.write\_string(**'G8'**, **'S.Y.:'**, rightBoldHeadless)  
 outputSheet.write\_string(**'H8'**, **'\_\_'** + schoolyear + **'\_\_'**, uleftHeadless)  
 *# ...Semester* outputSheet.write\_string(**'G9'**, **'Semester:'**, rightBoldHeadless)  
 outputSheet.write\_string(**'H9'**, **'\_'** + semester + **'\_'**, uleftHeadless)  
 *# ...bordercells* for row in range(12, 40):  
 for col in range(1, 8):  
 outputSheet.write(row, col, **''**, centerHead)  
 *# ...below table* outputSheet.merge\_range(**'A42:B42'**, **' OFFICIAL TIME:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'A45:B45'**, **' OVERLOAD:'**, leftBoldHeadless)  
 outputSheet.write\_string(**'C41'**, **'DAY'**, centerBoldHeadless)  
 outputSheet.write\_string(**'C42'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.write\_string(**'C43'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.write\_string(**'C44'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.write\_string(**'C45'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.write\_string(**'C46'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.write\_string(**'C47'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.merge\_range(**'E41:F41'**, **'TIME'**, centerBoldHeadless)  
 outputSheet.merge\_range(**'E42:F42'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.merge\_range(**'E43:F43'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.merge\_range(**'E44:F44'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.merge\_range(**'E45:F45'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.merge\_range(**'E46:F46'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.merge\_range(**'E47:F47'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 *# ...footer* outputSheet.write\_string(**'B48'**, **'Transaction ID'**, sheet\_details)  
 outputSheet.merge\_range(**'C48:G48'**, **''**, sheet\_details)  
 outputSheet.write\_string(**'B49'**, **'Signature'**, sheet\_details)  
 outputSheet.merge\_range(**'C49:G49'**, **''**, sheet\_details)  
  
 for MT in MT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in MT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 instructor\_sectionBlocks[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 instructor\_subjects[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 instructor\_rooms[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 MT\_timeBlocks[sheet][MT\_timeBlocks[sheet].index(MT)]), classSched)  
  
 for numb\_of\_instructors in range(0, len(instructors)):  
 sheet=numb\_of\_instructors  
 INSTRUCTORBLOCKS()  
 outWorkbook.close()  
  
 @staticmethod  
 def XSECTIONXLSX(schedule):  
 sections=dbMgr.get\_sections()  
 sectionCodes=list()  
 section\_instructorBlocks=list()  
 MT\_timeBlocks=list()  
 section\_rooms=list()  
 section\_subjects=list()  
 xsection\_instructorBlocks=list()  
 xMT\_timeBlocks=list()  
 xsection\_rooms=list()  
 xsection\_subjects=list()  
 ysection\_instructorBlocks=list()  
 yMT\_timeBlocks=list()  
 ysection\_rooms=list()  
 ysection\_subjects=list()  
 zsection\_instructorBlocks=list()  
 zMT\_timeBlocks=list()  
 zsection\_rooms=list()  
 zsection\_subjects=list()  
 for i in range(0, len(sections)):  
 sectionCodes.append(sections[i].get\_code())  
 instructorBlock=list()  
 timeBlock=list()  
 roomBlock=list()  
 subjectBlock=list()  
 xinstructorBlock=list()  
 xtimeBlock=list()  
 xroomBlock=list()  
 xsubjectBlock=list()  
 yinstructorBlock=list()  
 ytimeBlock=list()  
 yroomBlock=list()  
 ysubjectBlock=list()  
 zinstructorBlock=list()  
 ztimeBlock=list()  
 zroomBlock=list()  
 zsubjectBlock=list()  
 for j in range(0, len(schedule.get\_classes())):  
 if (schedule.get\_classes()[j].get\_section() == sections[i]):  
 instructorBlock.append(str(schedule.get\_classes()[j].get\_instructor().get\_number()))  
 timeBlock.append(str(schedule.get\_classes()[j].get\_meetingTime().get\_id()))  
 roomBlock.append(str(schedule.get\_classes()[j].get\_room().get\_number()))  
 subjectBlock.append(str(schedule.get\_classes()[j].get\_subject().get\_code()))  
 for j in range(0, len(schedule.get\_classes())):  
 if (schedule.get\_classes()[j].get\_subject().get\_numbHour() > schedule.get\_classes()[  
 j].get\_meetingTime().get\_MThour()):  
 if (schedule.get\_classes()[j].get\_section() == sections[i]):  
 instructorBlock.append(str(schedule.get\_classes()[j].get\_instructor().get\_number()))  
 timeBlock.append(str(schedule.get\_classes()[j].get\_meetingTime1().get\_id()))  
 roomBlock.append(str(schedule.get\_classes()[j].get\_room().get\_number()))  
 subjectBlock.append(str(schedule.get\_classes()[j].get\_subject().get\_code()))  
 for j in range(0, len(overtimeConflicts)):  
 if (overtimeConflicts[j].get\_section() == sections[i]):  
 xinstructorBlock.append(str(overtimeConflicts[j].get\_instructor().get\_number()))  
 xtimeBlock.append(str(overtimeConflicts[j].get\_meetingTime().get\_id()))  
 xroomBlock.append(str(overtimeConflicts[j].get\_room().get\_number()))  
 xsubjectBlock.append(str(overtimeConflicts[j].get\_subject().get\_code()))  
 for j in range(0, len(overtimeConflicts)):  
 if (overtimeConflicts[j].get\_subject().get\_numbHour() > overtimeConflicts[  
 j].get\_meetingTime().get\_MThour()):  
 if (overtimeConflicts[j].get\_section() == sections[i]):  
 xinstructorBlock.append(str(overtimeConflicts[j].get\_instructor().get\_number()))  
 xtimeBlock.append(str(overtimeConflicts[j].get\_meetingTime1().get\_id()))  
 xroomBlock.append(str(overtimeConflicts[j].get\_room().get\_number()))  
 xsubjectBlock.append(str(overtimeConflicts[j].get\_subject().get\_code()))  
 for j in range(0, len(overloadConflicts)):  
 if (overloadConflicts[j].get\_section() == sections[i]):  
 yinstructorBlock.append(str(overloadConflicts[j].get\_instructor().get\_number()))  
 ytimeBlock.append(str(overloadConflicts[j].get\_meetingTime().get\_id()))  
 yroomBlock.append(str(overloadConflicts[j].get\_room().get\_number()))  
 ysubjectBlock.append(str(overloadConflicts[j].get\_subject().get\_code()))  
 for j in range(0, len(overloadConflicts)):  
 if (overloadConflicts[j].get\_subject().get\_numbHour() > overloadConflicts[  
 j].get\_meetingTime().get\_MThour()):  
 if (overloadConflicts[j].get\_section() == sections[i]):  
 yinstructorBlock.append(str(overloadConflicts[j].get\_instructor().get\_number()))  
 ytimeBlock.append(str(overloadConflicts[j].get\_meetingTime1().get\_id()))  
 yroomBlock.append(str(overloadConflicts[j].get\_room().get\_number()))  
 ysubjectBlock.append(str(overloadConflicts[j].get\_subject().get\_code()))  
 for j in range(0, len(roombookingConflicts)):  
 if (roombookingConflicts[j].get\_section() == sections[i]):  
 zinstructorBlock.append(str(roombookingConflicts[j].get\_instructor().get\_number()))  
 ztimeBlock.append(str(roombookingConflicts[j].get\_meetingTime().get\_id()))  
 zroomBlock.append(str(roombookingConflicts[j].get\_room().get\_number()))  
 zsubjectBlock.append(str(roombookingConflicts[j].get\_subject().get\_code()))  
 for j in range(0, len(roombookingConflicts)):  
 if (roombookingConflicts[j].get\_subject().get\_numbHour() > roombookingConflicts[  
 j].get\_meetingTime().get\_MThour()):  
 if (roombookingConflicts[j].get\_section() == sections[i]):  
 zinstructorBlock.append(str(roombookingConflicts[j].get\_instructor().get\_number()))  
 ztimeBlock.append(str(roombookingConflicts[j].get\_meetingTime1().get\_id()))  
 zroomBlock.append(str(roombookingConflicts[j].get\_room().get\_number()))  
 zsubjectBlock.append(str(roombookingConflicts[j].get\_subject().get\_code()))  
 section\_instructorBlocks.append(instructorBlock)  
 MT\_timeBlocks.append(timeBlock)  
 section\_rooms.append(roomBlock)  
 section\_subjects.append(subjectBlock)  
 xsection\_instructorBlocks.append(xinstructorBlock)  
 xMT\_timeBlocks.append(xtimeBlock)  
 xsection\_rooms.append(xroomBlock)  
 xsection\_subjects.append(xsubjectBlock)  
 ysection\_instructorBlocks.append(yinstructorBlock)  
 yMT\_timeBlocks.append(ytimeBlock)  
 ysection\_rooms.append(yroomBlock)  
 ysection\_subjects.append(ysubjectBlock)  
 zsection\_instructorBlocks.append(zinstructorBlock)  
 zMT\_timeBlocks.append(ztimeBlock)  
 zsection\_rooms.append(zroomBlock)  
 zsection\_subjects.append(zsubjectBlock)  
 timeCells=dbMgr.get\_meetingTimes()  
 meeetingTimes=list()  
 cells=list()  
 for i in range(0, len(timeCells)):  
 meeetingTimes.append(timeCells[i].get\_id())  
 cells.append(timeCells[i].get\_cell())  
  
 *# create file (workbook) and worksheet* outWorkbook=xlsxwriter.Workbook(**"xsection.xlsx"**)  
  
 def SECTIONBLOCKS():  
 *# declare data* columnData=[**"Time"**, **"Monday"**, **"Tuesday"**, **"Wednesday"**, **"Thursday"**, **"Friday"**, **"Saturday"**, **"Sunday"**]  
 timeBlock=[**"7:00 - 7:30"**, **"7:30 - 8:00"**, **"8:00 - 8:30"**, **"8:30 - 9:00"**, **"9:00 - 9:30"**,  
 **"9:30 - 10:00"**,  
 **"10:00 - 10:30"**,  
 **"10:30 - 11:00"**, **"11:00 - 11:30"**, **"11:30 - 12:00"**, **"12:00 - 12:30"**, **"12:30 - 1:00"**,  
 **"1:00 - 1:30"**,  
 **"1:30 -2:00"**, **"2:00 - 2:30"**, **"2:30 - 3:00"**, **"3:00 - 3:30"**, **"3:30 - 4:00"**,  
 **"4:00 - 4:30"**,  
 **"4:30 - 5:00"**,  
 **"5:00 - 5:30"**, **"5:30 - 6:00"**, **"6:00 - 6:30"**, **"6:30 - 7:00"**, **"7:00 - 7:30"**,  
 **"7:30 - 8:00"**,  
 **"8:00 - 8:30"**,  
 **"8:30 - 9:00"**]  
 load\_title=**'STUDENT SCHEDULE'** summaryColumn=[**"SUBJECT"**, **"TIME"**, **"DAY"**, **"ROOM"**, **"UNIT"**, **"# OF STUDENTS"**, **"FACULTY"**]  
 outputSheet=outWorkbook.add\_worksheet(name = sectionCodes[sheet])  
 *# Increase the cell size of the merged cells to highlight the formatting.* outputSheet.set\_column(**'A:H'**, 15)  
 outputSheet.set\_default\_row(20)  
 *# Create a format to use in the...  
 # ...centerBoldHead.* centerBoldHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...centerBoldHeadless.* centerBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...leftBoldHeadless.* leftBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...uleftHeadless.* uleftHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...rightBoldHeadless.* rightBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 0,  
 **'align'**: **'right'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...centerHead.* centerHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...ucenterHeadless.* ucenterHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...icenterHeadless.* icenterHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'italic'**: 1,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...smallcenterHead.* smallcenterHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 7.5,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...xsmallcenterHead.* xsmallcenterHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 7.5,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'fadfca'**})  
 *# ...ysmallcenterHead.* ysmallcenterHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 7.5,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'ffe338'**})  
 *# ...zsmallcenterHead.* zsmallcenterHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 7.5,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'b8d4ff'**})  
 *# ...leftHeadless.* leftHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...details.* sheet\_details=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vbottom'**,  
 **'fg\_color'**: **'white'**})  
 *# ...header.* header=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 22,  
 **'font\_color'**: **'white'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'c31d3a'**}) *# Green, Accent 6, Darker 25%: 76933C  
 # ... columns.* column=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'white'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'c41e39'**}) *# Green, Accent 6: 9BBB59  
 # ...time.* time=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**}) *# Green, Accent 6, Lighter 40%: C4D798  
 # ...classSched.* classSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'ececec'**}) *# Emerald: 50C878  
 # ...xclassSched.* xclassSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'fadfca'**}) *# light 3 matte orange  
 # ...yclassSched.* yclassSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'ffe338'**})  
 *# ...zclassSched.* zclassSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'b8d4ff'**})  
 *# Write inside the Cell  
 # ...columnData* for item in range(len(columnData)):  
 outputSheet.write(11, item + 0, columnData[item], column)  
 *# ...timeBlock* for item in range(len(timeBlock)):  
 outputSheet.write(item + 12, 0, timeBlock[item], time)  
 *# ...Merge School* outputSheet.merge\_range(**'B1:F4'**, **''**, centerHead)  
 outputSheet.write\_rich\_string(**'B1'**, centerBoldHead, schooltext, centerHead, school\_infotext,  
 centerHead)  
 *# ...Load Title* outputSheet.merge\_range(**'B5:F6'**, load\_title, centerBoldHead)  
 *# ...VAA-OAP* outputSheet.merge\_range(**'A5:A6'**, vaa\_oap, centerHead)  
 *# ...Sheet Details* outputSheet.write\_string(**'G1'**, **'Index No.'**, sheet\_details)  
 outputSheet.write\_string(**'G2'**, **'Issue No.'**, sheet\_details)  
 outputSheet.write\_string(**'G3'**, **'Revision No.'**, sheet\_details)  
 outputSheet.write\_string(**'G4'**, **'Date'**, sheet\_details)  
 outputSheet.write\_string(**'G5'**, **'Page'**, sheet\_details)  
 outputSheet.write\_string(**'G6'**, **'QAC No.'**, sheet\_details)  
 outputSheet.write\_string(**'H1'**, str(indexno), sheet\_details)  
 outputSheet.write\_string(**'H2'**, str(issueno), sheet\_details)  
 outputSheet.write\_string(**'H3'**, str(revisionno), sheet\_details)  
 outputSheet.write\_string(**'H4'**, str(date), sheet\_details)  
 outputSheet.write\_string(**'H5'**, str(page), sheet\_details)  
 outputSheet.write\_string(**'H6'**, str(qacno), sheet\_details)  
 *# ...Logo* outputSheet.merge\_range(**'A1:A4'**, **''**, centerHead)  
 outputSheet.insert\_image(**'A1'**, **'tup-logo.png'**, {**'x\_scale'**: 1.1, **'y\_scale'**: 1.1})  
 *# ...College* outputSheet.merge\_range(**'A8:B8'**, **' COLLEGE:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C8:E8'**,  
 **'\_'** + college + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...Department* outputSheet.merge\_range(**'A9:B9'**, **' DEPARTMENT:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C9:E9'**,  
 **'\_'** + department + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...Name of Faculty* outputSheet.merge\_range(**'A10:B10'**, **' SECTION:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C10:E10'**,  
 **'\_'** + str(  
 sectionCodes[sheet]) + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...S.Y.* outputSheet.write\_string(**'G8'**, **'S.Y.:'**, rightBoldHeadless)  
 outputSheet.write\_string(**'H8'**, **'\_\_'** + schoolyear + **'\_\_'**, uleftHeadless)  
 *# ...Semester* outputSheet.write\_string(**'G9'**, **'Semester:'**, rightBoldHeadless)  
 outputSheet.write\_string(**'H9'**, **'\_'** + semester + **'\_'**, uleftHeadless)  
 *# ...bordercells* for row in range(12, 40):  
 for col in range(1, 8):  
 outputSheet.write(row, col, **''**, centerHead)  
 *# Summary Table  
 # ...summaryColumn* for item in range(len(summaryColumn)):  
 outputSheet.write(41, item, summaryColumn[item], column)  
 outputSheet.merge\_range(41, 6, 41, 7, summaryColumn[6], column)  
  
 for row in range(42, 54):  
 for col in range(0, 6):  
 outputSheet.write(row, col, **''**, centerHead)  
 outputSheet.merge\_range(row, 6, row, 7, **''**, centerHead)  
 classes=schedule.get\_classes()  
 class\_list=list()  
 overtimeConflicts\_list=list()  
 overloadConflicts\_list=list()  
 roombookingConflicts\_list=list()  
 for i in range(0, len(classes)):  
 if (classes[i].get\_section().get\_code() == sectionCodes[sheet]):  
 class\_list.append(classes[i])  
 for i in range(0, len(overtimeConflicts)):  
 if (overtimeConflicts[i].get\_section().get\_code() == sectionCodes[sheet]):  
 overtimeConflicts\_list.append(overtimeConflicts[i])  
 for i in range(0, len(overloadConflicts)):  
 if (overloadConflicts[i].get\_section().get\_code() == sectionCodes[sheet]):  
 overloadConflicts\_list.append(overloadConflicts[i])  
 for i in range(0, len(roombookingConflicts)):  
 if (roombookingConflicts[i].get\_section().get\_code() == sectionCodes[sheet]):  
 roombookingConflicts\_list.append(roombookingConflicts[i])  
 for i in range(0, len(class\_list)):  
 if class\_list[i].get\_section().get\_code() == sectionCodes[sheet]:  
 outputSheet.write(42 + i, 0, class\_list[i].get\_subject().get\_code(), smallcenterHead)  
 if class\_list[i].get\_meetingTime1().get\_id() != **'NULL'**:  
 outputSheet.write(42 + i, 1,  
 str(class\_list[i].get\_meetingTime().get\_time()) + **'**\n**'** + str(  
 class\_list[i].get\_meetingTime1().get\_time()), smallcenterHead)  
 outputSheet.write(42 + i, 2,  
 str(class\_list[i].get\_meetingTime().get\_day()[0:3]) + **'**\n**'** + str(  
 class\_list[i].get\_meetingTime1().get\_day()[0:3]), smallcenterHead)  
 else:  
 outputSheet.write(42 + i, 1, class\_list[i].get\_meetingTime().get\_time(),  
 smallcenterHead)  
 outputSheet.write(42 + i, 2, class\_list[i].get\_meetingTime().get\_day()[0:3],  
 smallcenterHead)  
 outputSheet.write(42 + i, 3, class\_list[i].get\_room().get\_number(), smallcenterHead)  
 outputSheet.write(42 + i, 4, class\_list[i].get\_subject().get\_units(), smallcenterHead)  
 outputSheet.write(42 + i, 5, class\_list[i].get\_subject().get\_maxNumbOfStudents(),  
 smallcenterHead)  
 outputSheet.write(42 + i, 6, class\_list[i].get\_instructor().get\_name(), smallcenterHead)  
 for i in range(0, len(overtimeConflicts\_list)):  
 if overtimeConflicts\_list[i].get\_section().get\_code() == sectionCodes[sheet]:  
 outputSheet.write(42 + i + len(class\_list), 0,  
 overtimeConflicts\_list[i].get\_subject().get\_code(), xsmallcenterHead)  
 if overtimeConflicts\_list[i].get\_meetingTime1().get\_id() != **'NULL'**:  
 outputSheet.write(42 + i + len(class\_list), 1, str(  
 overtimeConflicts\_list[i].get\_meetingTime().get\_time()) + **'**\n**'** + str(  
 overtimeConflicts\_list[i].get\_meetingTime1().get\_time()), xsmallcenterHead)  
 outputSheet.write(42 + i + len(class\_list), 2, str(  
 overtimeConflicts\_list[i].get\_meetingTime().get\_day()[0:3]) + **'**\n**'** + str(  
 overtimeConflicts\_list[i].get\_meetingTime1().get\_day()[0:3]), xsmallcenterHead)  
 else:  
 outputSheet.write(42 + i + len(class\_list), 1,  
 overtimeConflicts\_list[i].get\_meetingTime().get\_time(),  
 xsmallcenterHead)  
 outputSheet.write(42 + i + len(class\_list), 2,  
 overtimeConflicts\_list[i].get\_meetingTime().get\_day()[0:3],  
 xsmallcenterHead)  
 outputSheet.write(42 + i + len(class\_list), 3,  
 overtimeConflicts\_list[i].get\_room().get\_number(), xsmallcenterHead)  
 outputSheet.write(42 + i + len(class\_list), 4,  
 overtimeConflicts\_list[i].get\_subject().get\_units(), xsmallcenterHead)  
 outputSheet.write(42 + i + len(class\_list), 5,  
 overtimeConflicts\_list[i].get\_subject().get\_maxNumbOfStudents(),  
 xsmallcenterHead)  
 outputSheet.write(42 + i + len(class\_list), 6,  
 overtimeConflicts\_list[i].get\_instructor().get\_name(), xsmallcenterHead)  
 for i in range(0, len(overloadConflicts\_list)):  
 if overloadConflicts\_list[i].get\_section().get\_code() == sectionCodes[sheet]:  
 outputSheet.write(42 + i + len(class\_list) + len(overtimeConflicts\_list), 0,  
 overloadConflicts\_list[i].get\_subject().get\_code(), ysmallcenterHead)  
 if overloadConflicts\_list[i].get\_meetingTime1().get\_id() != **'NULL'**:  
 outputSheet.write(42 + i + len(class\_list) + len(overtimeConflicts\_list), 1,  
 str(overloadConflicts\_list[  
 i].get\_meetingTime().get\_time()) + **'**\n**'** + str(  
 overloadConflicts\_list[i].get\_meetingTime1().get\_time()),  
 ysmallcenterHead)  
 outputSheet.write(42 + i + len(class\_list) + len(overtimeConflicts\_list), 2, str(  
 overloadConflicts\_list[i].get\_meetingTime().get\_day()[0:3]) + **'**\n**'** + str(  
 overloadConflicts\_list[i].get\_meetingTime1().get\_day()[0:3]), ysmallcenterHead)  
 else:  
 outputSheet.write(42 + i + len(class\_list) + len(overtimeConflicts\_list), 1,  
 overloadConflicts\_list[i].get\_meetingTime().get\_time(),  
 ysmallcenterHead)  
 outputSheet.write(42 + i + len(class\_list) + len(overtimeConflicts\_list), 2,  
 overloadConflicts\_list[i].get\_meetingTime().get\_day()[0:3],  
 ysmallcenterHead)  
 outputSheet.write(42 + i + len(class\_list) + len(overtimeConflicts\_list), 3,  
 overloadConflicts\_list[i].get\_room().get\_number(), ysmallcenterHead)  
 outputSheet.write(42 + i + len(class\_list) + len(overtimeConflicts\_list), 4,  
 overloadConflicts\_list[i].get\_subject().get\_units(), ysmallcenterHead)  
 outputSheet.write(42 + i + len(class\_list) + len(overtimeConflicts\_list), 5,  
 overloadConflicts\_list[i].get\_subject().get\_maxNumbOfStudents(),  
 ysmallcenterHead)  
 outputSheet.write(42 + i + len(class\_list) + len(overtimeConflicts\_list), 6,  
 overloadConflicts\_list[i].get\_instructor().get\_name(), ysmallcenterHead)  
 for i in range(0, len(roombookingConflicts\_list)):  
 if roombookingConflicts\_list[i].get\_section().get\_code() == sectionCodes[sheet]:  
 outputSheet.write(42 + i + len(class\_list) + len(overtimeConflicts\_list), 0,  
 roombookingConflicts\_list[i].get\_subject().get\_code(), zsmallcenterHead)  
 if roombookingConflicts\_list[i].get\_meetingTime1().get\_id() != **'NULL'**:  
 outputSheet.write(42 + i + len(class\_list) + len(overtimeConflicts\_list), 1,  
 str(roombookingConflicts\_list[  
 i].get\_meetingTime().get\_time()) + **'**\n**'** + str(  
 roombookingConflicts\_list[i].get\_meetingTime1().get\_time()),  
 zsmallcenterHead)  
 outputSheet.write(42 + i + len(class\_list) + len(overtimeConflicts\_list), 2, str(  
 roombookingConflicts\_list[i].get\_meetingTime().get\_day()[0:3]) + **'**\n**'** + str(  
 roombookingConflicts\_list[i].get\_meetingTime1().get\_day()[0:3]), zsmallcenterHead)  
 else:  
 outputSheet.write(42 + i + len(class\_list) + len(overtimeConflicts\_list), 1,  
 roombookingConflicts\_list[i].get\_meetingTime().get\_time(),  
 zsmallcenterHead)  
 outputSheet.write(42 + i + len(class\_list) + len(overtimeConflicts\_list), 2,  
 roombookingConflicts\_list[i].get\_meetingTime().get\_day()[0:3],  
 zsmallcenterHead)  
 outputSheet.write(42 + i + len(class\_list) + len(overtimeConflicts\_list), 3,  
 roombookingConflicts\_list[i].get\_room().get\_number(), zsmallcenterHead)  
 outputSheet.write(42 + i + len(class\_list) + len(overtimeConflicts\_list), 4,  
 roombookingConflicts\_list[i].get\_subject().get\_units(), zsmallcenterHead)  
 outputSheet.write(42 + i + len(class\_list) + len(overtimeConflicts\_list), 5,  
 roombookingConflicts\_list[i].get\_subject().get\_maxNumbOfStudents(),  
 zsmallcenterHead)  
 outputSheet.write(42 + i + len(class\_list) + len(overtimeConflicts\_list), 6,  
 roombookingConflicts\_list[i].get\_instructor().get\_name(), zsmallcenterHead)  
 *# ...Authority Sign* outputSheet.merge\_range(**'A55:C55'**, **' Prepared By:'**, leftHeadless)  
 outputSheet.merge\_range(**'A57:C57'**, **'\_\_\_\_\_'** + str(depthead) + **'\_\_\_\_\_'**, ucenterHeadless)  
 outputSheet.merge\_range(**'A58:C58'**, **'Department Head'**, icenterHeadless)  
 outputSheet.merge\_range(**'F55:H55'**, **' Recommending Approval:'**, leftHeadless)  
 outputSheet.merge\_range(**'F57:H57'**, **'\_\_\_\_\_'** + str(dean) + **'\_\_\_\_\_'**, ucenterHeadless)  
 outputSheet.merge\_range(**'F58:H58'**, **'College Dean'**, icenterHeadless)  
 *# ...footer* outputSheet.write\_string(**'B60'**, **'Transaction ID'**, sheet\_details)  
 outputSheet.merge\_range(**'C60:G60'**, **''**, sheet\_details)  
 outputSheet.write\_string(**'B61'**, **'Signature'**, sheet\_details)  
 outputSheet.merge\_range(**'C61:G61'**, **''**, sheet\_details)  
  
 for MT in MT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in MT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 section\_instructorBlocks[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 section\_rooms[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 section\_subjects[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 MT\_timeBlocks[sheet][MT\_timeBlocks[sheet].index(MT)]), classSched)  
 for MT in xMT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in xMT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 xsection\_instructorBlocks[sheet][xMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 xsection\_rooms[sheet][xMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 xsection\_subjects[sheet][xMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 xMT\_timeBlocks[sheet][xMT\_timeBlocks[sheet].index(MT)]), xclassSched)  
 for MT in yMT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in yMT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 ysection\_instructorBlocks[sheet][yMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 ysection\_rooms[sheet][yMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 ysection\_subjects[sheet][yMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 yMT\_timeBlocks[sheet][yMT\_timeBlocks[sheet].index(MT)]), yclassSched)  
 for MT in zMT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in zMT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 zsection\_instructorBlocks[sheet][zMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 zsection\_rooms[sheet][zMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 zsection\_subjects[sheet][zMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 zMT\_timeBlocks[sheet][zMT\_timeBlocks[sheet].index(MT)]), zclassSched)  
  
 for numb\_of\_sections in range(0, len(sections)):  
 sheet=numb\_of\_sections  
 SECTIONBLOCKS()  
 outWorkbook.close()  
  
 @staticmethod  
 def XROOMXLSX(schedule):  
 rooms=dbMgr.get\_rooms()  
 roomNumbers=list()  
 room\_sectionBlocks=list()  
 MT\_timeBlocks=list()  
 room\_instructors=list()  
 room\_subjects=list()  
 xroom\_sectionBlocks=list()  
 xMT\_timeBlocks=list()  
 xroom\_instructors=list()  
 xroom\_subjects=list()  
 yroom\_sectionBlocks=list()  
 yMT\_timeBlocks=list()  
 yroom\_instructors=list()  
 yroom\_subjects=list()  
 zroom\_sectionBlocks=list()  
 zMT\_timeBlocks=list()  
 zroom\_instructors=list()  
 zroom\_subjects=list()  
 for i in range(0, len(rooms)):  
 roomNumbers.append(rooms[i].get\_number())  
 sectionBlock=list()  
 instructorBlock=list()  
 subjectBlock=list()  
 timeBlock=list()  
 xsectionBlock=list()  
 xinstructorBlock=list()  
 xsubjectBlock=list()  
 xtimeBlock=list()  
 ysectionBlock=list()  
 yinstructorBlock=list()  
 ysubjectBlock=list()  
 ytimeBlock=list()  
 zsectionBlock=list()  
 zinstructorBlock=list()  
 zsubjectBlock=list()  
 ztimeBlock=list()  
 for j in range(0, len(schedule.get\_classes())):  
 if (schedule.get\_classes()[j].get\_room() == rooms[i]):  
 sectionBlock.append(str(schedule.get\_classes()[j].get\_section().get\_code()))  
 instructorBlock.append(str(schedule.get\_classes()[j].get\_instructor().get\_number()))  
 subjectBlock.append(str(schedule.get\_classes()[j].get\_subject().get\_code()))  
 timeBlock.append(str(schedule.get\_classes()[j].get\_meetingTime().get\_id()))  
 for j in range(0, len(schedule.get\_classes())):  
 if (schedule.get\_classes()[j].get\_subject().get\_numbHour() > schedule.get\_classes()[  
 j].get\_meetingTime().get\_MThour()):  
 if (schedule.get\_classes()[j].get\_room() == rooms[i]):  
 sectionBlock.append(str(schedule.get\_classes()[j].get\_section().get\_code()))  
 instructorBlock.append(str(schedule.get\_classes()[j].get\_instructor().get\_number()))  
 subjectBlock.append(str(schedule.get\_classes()[j].get\_subject().get\_code()))  
 timeBlock.append(str(schedule.get\_classes()[j].get\_meetingTime1().get\_id()))  
 for j in range(0, len(overtimeConflicts)):  
 if (overtimeConflicts[j].get\_room() == rooms[i]):  
 xsectionBlock.append(overtimeConflicts[j].get\_section().get\_code())  
 xinstructorBlock.append(overtimeConflicts[j].get\_instructor().get\_number())  
 xsubjectBlock.append(overtimeConflicts[j].get\_subject().get\_code())  
 xtimeBlock.append(str(overtimeConflicts[j].get\_meetingTime().get\_id()))  
 for j in range(0, len(overtimeConflicts)):  
 if (overtimeConflicts[j].get\_subject().get\_numbHour() > overtimeConflicts[  
 j].get\_meetingTime().get\_MThour()):  
 if (overtimeConflicts[j].get\_room() == rooms[i]):  
 xsectionBlock.append(str(overtimeConflicts[j].get\_section().get\_code()))  
 xinstructorBlock.append(str(overtimeConflicts[j].get\_instructor().get\_number()))  
 xsubjectBlock.append(str(overtimeConflicts[j].get\_subject().get\_code()))  
 xtimeBlock.append(str(overtimeConflicts[j].get\_meetingTime1().get\_id()))  
 for j in range(0, len(overloadConflicts)):  
 if (overloadConflicts[j].get\_room() == rooms[i]):  
 ysectionBlock.append(overloadConflicts[j].get\_section().get\_code())  
 yinstructorBlock.append(overloadConflicts[j].get\_instructor().get\_number())  
 ysubjectBlock.append(overloadConflicts[j].get\_subject().get\_code())  
 ytimeBlock.append(str(overloadConflicts[j].get\_meetingTime().get\_id()))  
 for j in range(0, len(overloadConflicts)):  
 if (overloadConflicts[j].get\_subject().get\_numbHour() > overloadConflicts[  
 j].get\_meetingTime().get\_MThour()):  
 if (overloadConflicts[j].get\_room() == rooms[i]):  
 ysectionBlock.append(str(overloadConflicts[j].get\_section().get\_code()))  
 yinstructorBlock.append(str(overloadConflicts[j].get\_instructor().get\_number()))  
 ysubjectBlock.append(str(overloadConflicts[j].get\_subject().get\_code()))  
 ytimeBlock.append(str(overloadConflicts[j].get\_meetingTime1().get\_id()))  
 for j in range(0, len(roombookingConflicts)):  
 if (roombookingConflicts[j].get\_room() == rooms[i]):  
 zsectionBlock.append(roombookingConflicts[j].get\_section().get\_code())  
 zinstructorBlock.append(roombookingConflicts[j].get\_instructor().get\_number())  
 zsubjectBlock.append(roombookingConflicts[j].get\_subject().get\_code())  
 ztimeBlock.append(str(roombookingConflicts[j].get\_meetingTime().get\_id()))  
 for j in range(0, len(roombookingConflicts)):  
 if (roombookingConflicts[j].get\_subject().get\_numbHour() > roombookingConflicts[  
 j].get\_meetingTime().get\_MThour()):  
 if (roombookingConflicts[j].get\_room() == rooms[i]):  
 zsectionBlock.append(str(roombookingConflicts[j].get\_section().get\_code()))  
 zinstructorBlock.append(str(roombookingConflicts[j].get\_instructor().get\_number()))  
 zsubjectBlock.append(str(roombookingConflicts[j].get\_subject().get\_code()))  
 ztimeBlock.append(str(roombookingConflicts[j].get\_meetingTime1().get\_id()))  
 room\_sectionBlocks.append(sectionBlock)  
 room\_instructors.append(instructorBlock)  
 room\_subjects.append(subjectBlock)  
 MT\_timeBlocks.append(timeBlock)  
 xroom\_sectionBlocks.append(xsectionBlock)  
 xroom\_instructors.append(xinstructorBlock)  
 xroom\_subjects.append(xsubjectBlock)  
 xMT\_timeBlocks.append(xtimeBlock)  
 yroom\_sectionBlocks.append(ysectionBlock)  
 yroom\_instructors.append(yinstructorBlock)  
 yroom\_subjects.append(ysubjectBlock)  
 yMT\_timeBlocks.append(ytimeBlock)  
 zroom\_sectionBlocks.append(zsectionBlock)  
 zroom\_instructors.append(zinstructorBlock)  
 zroom\_subjects.append(zsubjectBlock)  
 zMT\_timeBlocks.append(ztimeBlock)  
 timeCells=dbMgr.get\_meetingTimes()  
 meeetingTimes=list()  
 cells=list()  
 for i in range(0, len(timeCells)):  
 meeetingTimes.append(timeCells[i].get\_id())  
 cells.append(timeCells[i].get\_cell())  
  
 *# create file (workbook) and worksheet* outWorkbook=xlsxwriter.Workbook(**"xroom.xlsx"**)  
  
 def ROOMBLOCKS():  
 *# declare data* columnData=[**"Time"**, **"Monday"**, **"Tuesday"**, **"Wednesday"**, **"Thursday"**, **"Friday"**, **"Saturday"**, **"Sunday"**]  
 timeBlock=[**"7:00 - 7:30"**, **"7:30 - 8:00"**, **"8:00 - 8:30"**, **"8:30 - 9:00"**, **"9:00 - 9:30"**,  
 **"9:30 - 10:00"**,  
 **"10:00 - 10:30"**,  
 **"10:30 - 11:00"**, **"11:00 - 11:30"**, **"11:30 - 12:00"**, **"12:00 - 12:30"**, **"12:30 - 1:00"**,  
 **"1:00 - 1:30"**,  
 **"1:30 -2:00"**, **"2:00 - 2:30"**, **"2:30 - 3:00"**, **"3:00 - 3:30"**, **"3:30 - 4:00"**,  
 **"4:00 - 4:30"**,  
 **"4:30 - 5:00"**,  
 **"5:00 - 5:30"**, **"5:30 - 6:00"**, **"6:00 - 6:30"**, **"6:30 - 7:00"**, **"7:00 - 7:30"**,  
 **"7:30 - 8:00"**,  
 **"8:00 - 8:30"**,  
 **"8:30 - 9:00"**]  
 load\_title=**'ROOM SCHEDULE'** outputSheet=outWorkbook.add\_worksheet(name = roomNumbers[sheet])  
 *# Increase the cell size of the merged cells to highlight the formatting.* outputSheet.set\_column(**'A:H'**, 15)  
 outputSheet.set\_default\_row(20)  
 *# Create a format to use in the...  
 # ...centerBoldHead.* centerBoldHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...centerBoldHeadless.* centerBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...leftBoldHeadless.* leftBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...uleftHeadless.* uleftHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...rightBoldHeadless.* rightBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 0,  
 **'align'**: **'right'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...centerHead.* centerHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...ucenterHeadless.* ucenterHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...icenterHeadless.* icenterHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'italic'**: 1,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...smallcenterHead.* smallcenterHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 7.5,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...leftHeadless.* leftHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...details.* sheet\_details=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vbottom'**,  
 **'fg\_color'**: **'white'**})  
 *# ...header.* header=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 22,  
 **'font\_color'**: **'white'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'c31d3a'**}) *# Green, Accent 6, Darker 25%: 76933C  
 # ... columns.* column=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'white'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'c41e39'**}) *# Green, Accent 6: 9BBB59  
 # ...time.* time=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**}) *# Green, Accent 6, Lighter 40%: C4D798  
 # ...classSched.* classSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'ececec'**}) *# Emerald: 50C878  
 # ...xclassSched.* xclassSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'fadfca'**}) *# light 3 matte orange  
 # ...yclassSched.* yclassSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'ffe338'**}) *# Banana Yellow  
 # ...zclassSched.* zclassSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'b8d4ff'**})  
 *# Write inside the Cell  
 # ...columnData* for item in range(len(columnData)):  
 outputSheet.write(11, item + 0, columnData[item], column)  
 *# ...timeBlock* for item in range(len(timeBlock)):  
 outputSheet.write(item + 12, 0, timeBlock[item], time)  
 *# ...Merge School* outputSheet.merge\_range(**'B1:F4'**, **''**, centerHead)  
 outputSheet.write\_rich\_string(**'B1'**, centerBoldHead, schooltext, centerHead, school\_infotext,  
 centerHead)  
 *# ...Load Title* outputSheet.merge\_range(**'B5:F6'**, load\_title, centerBoldHead)  
 *# ...VAA-OAP* outputSheet.merge\_range(**'A5:A6'**, vaa\_oap, centerHead)  
 *# ...Sheet Details* outputSheet.write\_string(**'G1'**, **'Index No.'**, sheet\_details)  
 outputSheet.write\_string(**'G2'**, **'Issue No.'**, sheet\_details)  
 outputSheet.write\_string(**'G3'**, **'Revision No.'**, sheet\_details)  
 outputSheet.write\_string(**'G4'**, **'Date'**, sheet\_details)  
 outputSheet.write\_string(**'G5'**, **'Page'**, sheet\_details)  
 outputSheet.write\_string(**'G6'**, **'QAC No.'**, sheet\_details)  
 outputSheet.write\_string(**'H1'**, str(indexno), sheet\_details)  
 outputSheet.write\_string(**'H2'**, str(issueno), sheet\_details)  
 outputSheet.write\_string(**'H3'**, str(revisionno), sheet\_details)  
 outputSheet.write\_string(**'H4'**, str(date), sheet\_details)  
 outputSheet.write\_string(**'H5'**, str(page), sheet\_details)  
 outputSheet.write\_string(**'H6'**, str(qacno), sheet\_details)  
 *# ...Logo* outputSheet.merge\_range(**'A1:A4'**, **''**, centerHead)  
 outputSheet.insert\_image(**'A1'**, **'tup-logo.png'**, {**'x\_scale'**: 1.1, **'y\_scale'**: 1.1})  
 *# ...College* outputSheet.merge\_range(**'A8:B8'**, **' COLLEGE:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C8:E8'**,  
 **'\_'** + college + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...Department* outputSheet.merge\_range(**'A9:B9'**, **' DEPARTMENT:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C9:E9'**,  
 **'\_'** + department + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...Name of Faculty* outputSheet.merge\_range(**'A10:B10'**, **' ROOM:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C10:E10'**,  
 **'\_'** + str(  
 roomNumbers[sheet]) + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...S.Y.* outputSheet.write\_string(**'G8'**, **'S.Y.:'**, rightBoldHeadless)  
 outputSheet.write\_string(**'H8'**, **'\_\_'** + schoolyear + **'\_\_'**, uleftHeadless)  
 *# ...Semester* outputSheet.write\_string(**'G9'**, **'Semester:'**, rightBoldHeadless)  
 outputSheet.write\_string(**'H9'**, **'\_'** + semester + **'\_'**, uleftHeadless)  
 *# ...bordercells* for row in range(12, 40):  
 for col in range(1, 8):  
 outputSheet.write(row, col, **''**, centerHead)  
 *# ...Authority Sign* outputSheet.merge\_range(**'A42:C42'**, **' Prepared By:'**, leftHeadless)  
 outputSheet.merge\_range(**'A44:C44'**, **'\_\_\_\_\_'** + str(depthead) + **'\_\_\_\_\_'**, ucenterHeadless)  
 outputSheet.merge\_range(**'A45:C45'**, **'Department Head'**, icenterHeadless)  
 outputSheet.merge\_range(**'F42:H42'**, **' Recommending Approval:'**, leftHeadless)  
 outputSheet.merge\_range(**'F44:H44'**, **'\_\_\_\_\_'** + str(dean) + **'\_\_\_\_\_'**, ucenterHeadless)  
 outputSheet.merge\_range(**'F45:H45'**, **'College Dean'**, icenterHeadless)  
 *# ...footer* outputSheet.write\_string(**'B48'**, **'Transaction ID'**, sheet\_details)  
 outputSheet.merge\_range(**'C48:G48'**, **''**, sheet\_details)  
 outputSheet.write\_string(**'B49'**, **'Signature'**, sheet\_details)  
 outputSheet.merge\_range(**'C49:G49'**, **''**, sheet\_details)  
  
 for MT in MT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in MT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 room\_instructors[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 room\_sectionBlocks[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 room\_subjects[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 MT\_timeBlocks[sheet][MT\_timeBlocks[sheet].index(MT)]), classSched)  
 for MT in xMT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in xMT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 xroom\_instructors[sheet][xMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 xroom\_sectionBlocks[sheet][xMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 xroom\_subjects[sheet][xMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 xMT\_timeBlocks[sheet][xMT\_timeBlocks[sheet].index(MT)]), xclassSched)  
 for MT in yMT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in yMT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 yroom\_instructors[sheet][yMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 yroom\_sectionBlocks[sheet][yMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 yroom\_subjects[sheet][yMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 yMT\_timeBlocks[sheet][yMT\_timeBlocks[sheet].index(MT)]), yclassSched)  
 for MT in zMT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in zMT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 zroom\_instructors[sheet][zMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 zroom\_sectionBlocks[sheet][zMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 zroom\_subjects[sheet][zMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 zMT\_timeBlocks[sheet][zMT\_timeBlocks[sheet].index(MT)]), zclassSched)  
  
 for numb\_of\_rooms in range(0, len(rooms)):  
 sheet=numb\_of\_rooms  
 ROOMBLOCKS()  
 outWorkbook.close()  
  
 @staticmethod  
 def XINSTRUCTORXLSX(schedule):  
 instructors=dbMgr.get\_instructors()  
 instructorIDs=list()  
 instructor\_sectionBlocks=list()  
 MT\_timeBlocks=list()  
 instructor\_rooms=list()  
 instructor\_subjects=list()  
 xinstructor\_sectionBlocks=list()  
 xMT\_timeBlocks=list()  
 xinstructor\_rooms=list()  
 xinstructor\_subjects=list()  
 yinstructor\_sectionBlocks=list()  
 yMT\_timeBlocks=list()  
 yinstructor\_rooms=list()  
 yinstructor\_subjects=list()  
 zinstructor\_sectionBlocks=list()  
 zMT\_timeBlocks=list()  
 zinstructor\_rooms=list()  
 zinstructor\_subjects=list()  
 for i in range(0, len(instructors)):  
 instructorIDs.append(instructors[i].get\_name())  
 sectionBlock=list()  
 timeBlock=list()  
 roomBlock=list()  
 subjectBlock=list()  
 xsectionBlock=list()  
 xtimeBlock=list()  
 xroomBlock=list()  
 xsubjectBlock=list()  
 ysectionBlock=list()  
 ytimeBlock=list()  
 yroomBlock=list()  
 ysubjectBlock=list()  
 zsectionBlock=list()  
 ztimeBlock=list()  
 zroomBlock=list()  
 zsubjectBlock=list()  
 for j in range(0, len(schedule.get\_classes())):  
 if (schedule.get\_classes()[j].get\_instructor() == instructors[i]):  
 sectionBlock.append(str(schedule.get\_classes()[j].get\_section().get\_code()))  
 timeBlock.append(str(schedule.get\_classes()[j].get\_meetingTime().get\_id()))  
 roomBlock.append(str(schedule.get\_classes()[j].get\_room().get\_number()))  
 subjectBlock.append(str(schedule.get\_classes()[j].get\_subject().get\_code()))  
 for j in range(0, len(schedule.get\_classes())):  
 if (schedule.get\_classes()[j].get\_subject().get\_numbHour() > schedule.get\_classes()[  
 j].get\_meetingTime().get\_MThour()):  
 if (schedule.get\_classes()[j].get\_instructor() == instructors[i]):  
 sectionBlock.append(str(schedule.get\_classes()[j].get\_section().get\_code()))  
 timeBlock.append(str(schedule.get\_classes()[j].get\_meetingTime1().get\_id()))  
 roomBlock.append(str(schedule.get\_classes()[j].get\_room().get\_number()))  
 subjectBlock.append(str(schedule.get\_classes()[j].get\_subject().get\_code()))  
 for j in range(0, len(overtimeConflicts)):  
 if (overtimeConflicts[j].get\_instructor() == instructors[i]):  
 xsectionBlock.append(str(overtimeConflicts[j].get\_section().get\_code()))  
 xtimeBlock.append(str(overtimeConflicts[j].get\_meetingTime().get\_id()))  
 xroomBlock.append(str(overtimeConflicts[j].get\_room().get\_number()))  
 xsubjectBlock.append(str(overtimeConflicts[j].get\_subject().get\_code()))  
 for j in range(0, len(overtimeConflicts)):  
 if (overtimeConflicts[j].get\_subject().get\_numbHour() > overtimeConflicts[  
 j].get\_meetingTime().get\_MThour()):  
 if (overtimeConflicts[j].get\_instructor() == instructors[i]):  
 xsectionBlock.append(overtimeConflicts[j].get\_section().get\_code())  
 xtimeBlock.append(str(overtimeConflicts[j].get\_meetingTime1().get\_id()))  
 xroomBlock.append(str(overtimeConflicts[j].get\_room().get\_number()))  
 xsubjectBlock.append(str(overtimeConflicts[j].get\_subject().get\_code()))  
 for j in range(0, len(overloadConflicts)):  
 if (overloadConflicts[j].get\_instructor() == instructors[i]):  
 ysectionBlock.append(str(overloadConflicts[j].get\_section().get\_code()))  
 ytimeBlock.append(str(overloadConflicts[j].get\_meetingTime().get\_id()))  
 yroomBlock.append(str(overloadConflicts[j].get\_room().get\_number()))  
 ysubjectBlock.append(str(overloadConflicts[j].get\_subject().get\_code()))  
 for j in range(0, len(overloadConflicts)):  
 if (overloadConflicts[j].get\_subject().get\_numbHour() > overloadConflicts[  
 j].get\_meetingTime().get\_MThour()):  
 if (overloadConflicts[j].get\_instructor() == instructors[i]):  
 ysectionBlock.append(overloadConflicts[j].get\_section().get\_code())  
 ytimeBlock.append(str(overloadConflicts[j].get\_meetingTime1().get\_id()))  
 yroomBlock.append(str(overloadConflicts[j].get\_room().get\_number()))  
 ysubjectBlock.append(str(overloadConflicts[j].get\_subject().get\_code()))  
 for j in range(0, len(roombookingConflicts)):  
 if (roombookingConflicts[j].get\_instructor() == instructors[i]):  
 zsectionBlock.append(str(roombookingConflicts[j].get\_section().get\_code()))  
 ztimeBlock.append(str(roombookingConflicts[j].get\_meetingTime().get\_id()))  
 zroomBlock.append(str(roombookingConflicts[j].get\_room().get\_number()))  
 zsubjectBlock.append(str(roombookingConflicts[j].get\_subject().get\_code()))  
 for j in range(0, len(roombookingConflicts)):  
 if (roombookingConflicts[j].get\_subject().get\_numbHour() > roombookingConflicts[  
 j].get\_meetingTime().get\_MThour()):  
 if (roombookingConflicts[j].get\_instructor() == instructors[i]):  
 zsectionBlock.append(roombookingConflicts[j].get\_section().get\_code())  
 ztimeBlock.append(str(roombookingConflicts[j].get\_meetingTime1().get\_id()))  
 zroomBlock.append(str(roombookingConflicts[j].get\_room().get\_number()))  
 zsubjectBlock.append(str(roombookingConflicts[j].get\_subject().get\_code()))  
 instructor\_sectionBlocks.append(sectionBlock)  
 MT\_timeBlocks.append(timeBlock)  
 instructor\_rooms.append(roomBlock)  
 instructor\_subjects.append(subjectBlock)  
 xinstructor\_sectionBlocks.append(xsectionBlock)  
 xMT\_timeBlocks.append(xtimeBlock)  
 xinstructor\_rooms.append(xroomBlock)  
 xinstructor\_subjects.append(xsubjectBlock)  
 yinstructor\_sectionBlocks.append(ysectionBlock)  
 yMT\_timeBlocks.append(ytimeBlock)  
 yinstructor\_rooms.append(yroomBlock)  
 yinstructor\_subjects.append(ysubjectBlock)  
 zinstructor\_sectionBlocks.append(zsectionBlock)  
 zMT\_timeBlocks.append(ztimeBlock)  
 zinstructor\_rooms.append(zroomBlock)  
 zinstructor\_subjects.append(zsubjectBlock)  
 timeCells=dbMgr.get\_meetingTimes()  
 meeetingTimes=list()  
 cells=list()  
 for i in range(0, len(timeCells)):  
 meeetingTimes.append(timeCells[i].get\_id())  
 cells.append(timeCells[i].get\_cell())  
  
 *# create file (workbook) and worksheet* outWorkbook=xlsxwriter.Workbook(**"xinstructor.xlsx"**)  
  
 def INSTRUCTORBLOCKS():  
 *# declare data* columnData=[**"Time"**, **"Monday"**, **"Tuesday"**, **"Wednesday"**, **"Thursday"**, **"Friday"**, **"Saturday"**, **"Sunday"**]  
 timeBlock=[**"7:00 - 7:30"**, **"7:30 - 8:00"**, **"8:00 - 8:30"**, **"8:30 - 9:00"**, **"9:00 - 9:30"**,  
 **"9:30 - 10:00"**,  
 **"10:00 - 10:30"**,  
 **"10:30 - 11:00"**, **"11:00 - 11:30"**, **"11:30 - 12:00"**, **"12:00 - 12:30"**, **"12:30 - 1:00"**,  
 **"1:00 - 1:30"**,  
 **"1:30 -2:00"**, **"2:00 - 2:30"**, **"2:30 - 3:00"**, **"3:00 - 3:30"**, **"3:30 - 4:00"**,  
 **"4:00 - 4:30"**,  
 **"4:30 - 5:00"**,  
 **"5:00 - 5:30"**, **"5:30 - 6:00"**, **"6:00 - 6:30"**, **"6:30 - 7:00"**, **"7:00 - 7:30"**,  
 **"7:30 - 8:00"**,  
 **"8:00 - 8:30"**,  
 **"8:30 - 9:00"**]  
 load\_title=**'INDIVIDUAL FACULTY LOADING'** outputSheet=outWorkbook.add\_worksheet(name = instructorIDs[sheet])  
 *# Increase the cell size of the merged cells to highlight the formatting.* outputSheet.set\_column(**'A:H'**, 15)  
 outputSheet.set\_default\_row(20)  
 *# Create a format to use in the...  
 # ...centerBoldHead.* centerBoldHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...centerBoldHeadless.* centerBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...leftBoldHeadless.* leftBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...uleftHeadless.* uleftHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...rightBoldHeadless.* rightBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 0,  
 **'align'**: **'right'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...centerHead.* centerHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...ucenterHeadless.* ucenterHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...icenterHeadless.* icenterHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'italic'**: 1,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...smallcenterHead.* smallcenterHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 7.5,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...leftHeadless.* leftHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...details.* sheet\_details=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vbottom'**,  
 **'fg\_color'**: **'white'**})  
 *# ...header.* header=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 22,  
 **'font\_color'**: **'white'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'c31d3a'**}) *# Green, Accent 6, Darker 25%: 76933C  
 # ... columns.* column=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'white'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'c41e39'**}) *# Green, Accent 6: 9BBB59  
 # ...time.* time=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**}) *# Green, Accent 6, Lighter 40%: C4D798  
 # ...classSched.* classSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'ececec'**}) *# Emerald: 50C878  
 # ...xclassSched.* xclassSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'fadfca'**}) *# light 3 matte orange  
 # ...yclassSched.* yclassSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'ffe338'**}) *# Banana Yellow  
 # ...zclassSched.* zclassSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'b8d4ff'**})  
 *# Write inside the Cell  
 # ...columnData* for item in range(len(columnData)):  
 outputSheet.write(11, item + 0, columnData[item], column)  
 *# ...timeBlock* for item in range(len(timeBlock)):  
 outputSheet.write(item + 12, 0, timeBlock[item], time)  
 *# ...Merge School* outputSheet.merge\_range(**'B1:F4'**, **''**, centerHead)  
 outputSheet.write\_rich\_string(**'B1'**, centerBoldHead, schooltext, centerHead, school\_infotext,  
 centerHead)  
 *# ...Load Title* outputSheet.merge\_range(**'B5:F6'**, load\_title, centerBoldHead)  
 *# ...VAA-OAP* outputSheet.merge\_range(**'A5:A6'**, vaa\_oap, centerHead)  
 *# ...Sheet Details* outputSheet.write\_string(**'G1'**, **'Index No.'**, sheet\_details)  
 outputSheet.write\_string(**'G2'**, **'Issue No.'**, sheet\_details)  
 outputSheet.write\_string(**'G3'**, **'Revision No.'**, sheet\_details)  
 outputSheet.write\_string(**'G4'**, **'Date'**, sheet\_details)  
 outputSheet.write\_string(**'G5'**, **'Page'**, sheet\_details)  
 outputSheet.write\_string(**'G6'**, **'QAC No.'**, sheet\_details)  
 outputSheet.write\_string(**'H1'**, str(indexno), sheet\_details)  
 outputSheet.write\_string(**'H2'**, str(issueno), sheet\_details)  
 outputSheet.write\_string(**'H3'**, str(revisionno), sheet\_details)  
 outputSheet.write\_string(**'H4'**, str(date), sheet\_details)  
 outputSheet.write\_string(**'H5'**, str(page), sheet\_details)  
 outputSheet.write\_string(**'H6'**, str(qacno), sheet\_details)  
 *# ...Logo* outputSheet.merge\_range(**'A1:A4'**, **''**, centerHead)  
 outputSheet.insert\_image(**'A1'**, **'tup-logo.png'**, {**'x\_scale'**: 1.1, **'y\_scale'**: 1.1})  
 *# ...College* outputSheet.merge\_range(**'A8:B8'**, **' COLLEGE:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C8:E8'**,  
 **'\_'** + college + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...Department* outputSheet.merge\_range(**'A9:B9'**, **' DEPARTMENT:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C9:E9'**,  
 **'\_'** + department + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...Name of Faculty* outputSheet.merge\_range(**'A10:B10'**, **' NAME OF FACULTY:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C10:E10'**,  
 **'\_'** + str(  
 instructorIDs[sheet]) + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...S.Y.* outputSheet.write\_string(**'G8'**, **'S.Y.:'**, rightBoldHeadless)  
 outputSheet.write\_string(**'H8'**, **'\_\_'** + schoolyear + **'\_\_'**, uleftHeadless)  
 *# ...Semester* outputSheet.write\_string(**'G9'**, **'Semester:'**, rightBoldHeadless)  
 outputSheet.write\_string(**'H9'**, **'\_'** + semester + **'\_'**, uleftHeadless)  
 *# ...bordercells* for row in range(12, 40):  
 for col in range(1, 8):  
 outputSheet.write(row, col, **''**, centerHead)  
 *# ...below table* outputSheet.merge\_range(**'A42:B42'**, **' OFFICIAL TIME:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'A45:B45'**, **' OVERLOAD:'**, leftBoldHeadless)  
 outputSheet.write\_string(**'C41'**, **'DAY'**, centerBoldHeadless)  
 outputSheet.write\_string(**'C42'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.write\_string(**'C43'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.write\_string(**'C44'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.write\_string(**'C45'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.write\_string(**'C46'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.write\_string(**'C47'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.merge\_range(**'E41:F41'**, **'TIME'**, centerBoldHeadless)  
 outputSheet.merge\_range(**'E42:F42'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.merge\_range(**'E43:F43'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.merge\_range(**'E44:F44'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.merge\_range(**'E45:F45'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.merge\_range(**'E46:F46'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.merge\_range(**'E47:F47'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 *# ...footer* outputSheet.write\_string(**'B48'**, **'Transaction ID'**, sheet\_details)  
 outputSheet.merge\_range(**'C48:G48'**, **''**, sheet\_details)  
 outputSheet.write\_string(**'B49'**, **'Signature'**, sheet\_details)  
 outputSheet.merge\_range(**'C49:G49'**, **''**, sheet\_details)  
  
 for MT in MT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in MT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 instructor\_sectionBlocks[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 instructor\_subjects[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 instructor\_rooms[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 MT\_timeBlocks[sheet][MT\_timeBlocks[sheet].index(MT)]), classSched)  
 for MT in xMT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in xMT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 xinstructor\_sectionBlocks[sheet][xMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 xinstructor\_subjects[sheet][xMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 xinstructor\_rooms[sheet][xMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 xMT\_timeBlocks[sheet][xMT\_timeBlocks[sheet].index(MT)]), xclassSched)  
 for MT in yMT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in yMT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 yinstructor\_sectionBlocks[sheet][yMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 yinstructor\_subjects[sheet][yMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 yinstructor\_rooms[sheet][yMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 yMT\_timeBlocks[sheet][yMT\_timeBlocks[sheet].index(MT)]), yclassSched)  
 for MT in zMT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in zMT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 zinstructor\_sectionBlocks[sheet][zMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 zinstructor\_subjects[sheet][zMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 zinstructor\_rooms[sheet][zMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 zMT\_timeBlocks[sheet][zMT\_timeBlocks[sheet].index(MT)]), zclassSched)  
  
 for numb\_of\_instructors in range(0, len(instructors)):  
 sheet=numb\_of\_instructors  
 INSTRUCTORBLOCKS()  
 outWorkbook.close()  
  
 @staticmethod  
 def display\_schedule\_meetingTimes(schedule):  
 print(**"> from 'meeting time' perspective"**)  
 meetingTimesTable=prettytable.PrettyTable(  
 [**'id'**, **'meeting time'**, **'classes [section,class,room,instructor,meeting-time] '**])  
 meetingTimes=dbMgr.get\_meetingTimes()  
 for i in range(0, len(meetingTimes)):  
 classes=list()  
 for j in range(0, len(schedule.get\_classes())):  
 if schedule.get\_classes()[j].get\_meetingTime() == meetingTimes[i]:  
 classes.append(str(schedule.get\_classes()[j]))  
 meetingTimesTable.add\_row([meetingTimes[i].get\_id(), meetingTimes[i].get\_time(), str(classes)])  
 print(meetingTimesTable)  
  
 @staticmethod  
 def display\_schedule\_rooms(schedule):  
 print(**"> from 'room' perspective"**)  
 scheduleRoomsTable=prettytable.PrettyTable(  
 [**'room'**, **'classes [section,class,room,instructor,meeting-time] '**])  
 rooms=dbMgr.get\_rooms()  
 for i in range(0, len(rooms)):  
 roomSchedule=list()  
 for j in range(0, len(schedule.get\_classes())):  
 if schedule.get\_classes()[j].get\_room() == rooms[i]:  
 roomSchedule.append(str(schedule.get\_classes()[j]))  
 scheduleRoomsTable.add\_row([str(rooms[i].get\_number()), str(roomSchedule)])  
 print(scheduleRoomsTable)  
  
 @staticmethod  
 def display\_schedule\_instructors(schedule):  
 print(**"> from 'instructor' perspective"**)  
 instructorsTable=prettytable.PrettyTable(  
 [**'ID'**, **'Instructor'**, **"Classes [section,class,room,instructor,meeting-time]"**,  
 **'Remaining Availability'**])  
 instructors=dbMgr.get\_instructors()  
 for i in range(0, len(instructors)):  
 availability=[]  
 for j in range(0, len(instructors[i].get\_availability())):  
 availability.append(instructors[i].get\_availability()[j])  
 classSchedule=list()  
 for j in range(0, len(schedule.get\_classes())):  
 if schedule.get\_classes()[j].get\_instructor() == instructors[i]:  
 classSchedule.append(str(schedule.get\_classes()[j]))  
 instructorsTable.add\_row(  
 [instructors[i].get\_number(), instructors[i].get\_name(), str(classSchedule), availability])  
 print(instructorsTable)  
  
 @staticmethod  
 def display\_schedule\_conflicts(schedule):  
 conflictsTable=prettytable.PrettyTable([**'conflict type'**, **'conflict between classes'**])  
 conflicts=schedule.get\_conflicts()  
 for i in range(0, len(conflicts)):  
 conflictsTable.add\_row([str(conflicts[i].get\_conflictType()),  
 str(**" and "**.join(map(str, conflicts[i].get\_conflictBetweenClasses())))])  
 if (len(conflicts) > 0): print(conflictsTable)  
  
 def find\_fittest\_schedule(VERBOSE\_FLAG):  
 sys.stdout=StdoutRedirector(textout)  
 generationNumber=0  
 finish=time.perf\_counter()  
 interval=round(finish - start, 2)  
 population=Population(POPULATION\_SIZE)  
 population.get\_schedules().sort(key = lambda x: x.get\_fitness(), reverse = True)  
 if (VERBOSE\_FLAG):  
 print(**"**\n**> Generation # "** + str(generationNumber) + **' @ Time Interval: '** + str(interval))  
 DisplayMgr.display\_schedule\_as\_table(population.get\_schedules()[0])  
 print(**"**\n**> Generation # "** + str(generationNumber) + **' @ Time Interval: '** + str(interval))  
 DisplayMgr.display\_schedule\_conflicts(population.get\_schedules()[0])  
 print(**"**\n**> Generation # "** + str(generationNumber) + **' @ Time Interval: '** + str(interval))  
 DisplayMgr.display\_generation(population)  
 geneticAlgorithm=GeneticAlgorithm()  
 while (population.get\_schedules()[0].get\_fitness() != 1.0):  
 finish=time.perf\_counter()  
 interval=round(finish - start, 2)  
 generationNumber+=1  
 population=geneticAlgorithm.evolve(population)  
 population.get\_schedules().sort(key = lambda x: x.get\_fitness(), reverse = True)  
 if (VERBOSE\_FLAG):  
 print(**"**\n**> Generation # "** + str(generationNumber) + **' @ Time Interval: '** + str(interval))  
 DisplayMgr.display\_schedule\_as\_table(population.get\_schedules()[0])  
 print(**"**\n**> Generation # "** + str(generationNumber) + **' @ Time Interval: '** + str(interval))  
 DisplayMgr.display\_schedule\_conflicts(population.get\_schedules()[0])  
 print(**"**\n**> Generation # "** + str(generationNumber) + **' @ Time Interval: '** + str(interval))  
 DisplayMgr.display\_generation(population)  
 if ((interval % (RECURRING\_UPDATE\_TIME\*3600)) < 1):  
 print(**"**\n**> Generation # "** + str(generationNumber) + **' @ Time Interval: '** + str(interval))  
 DisplayMgr.display\_schedule\_as\_table(population.get\_schedules()[0])  
 print(**"**\n**> Generation # "** + str(generationNumber) + **' @ Time Interval: '** + str(interval))  
 DisplayMgr.display\_schedule\_conflicts(population.get\_schedules()[0])  
 print(**"**\n**> Generation # "** + str(generationNumber) + **' @ Time Interval: '** + str(interval))  
 DisplayMgr.display\_generation(population)  
 for i in range(3):  
 *# frequencu is in Hz* freq=1500  
 *# duration is in milliseconds* dur=500  
 winsound.Beep(freq, dur)  
 if (interval >= (SCHEDULING\_TIME\_LIMIT\*3600)):  
 sys.stdout=StdoutRedirector(textout)  
 print(**"**\n**> Generation # "** + str(generationNumber) + **' @ Time Interval: '** + str(interval))  
 DisplayMgr.display\_schedule\_as\_table(population.get\_schedules()[0])  
 print(**"**\n**> Generation # "** + str(generationNumber) + **' @ Time Interval: '** + str(interval))  
 DisplayMgr.display\_schedule\_conflicts(population.get\_schedules()[0])  
 print(**"**\n**> Generation # "** + str(generationNumber) + **' @ Time Interval: '** + str(interval))  
 DisplayMgr.display\_generation(population)  
 break  
 if (population.get\_schedules()[0].get\_fitness() == 1.0):  
 sys.stdout=open(ARASTDOUT, **'a'**)  
 print(**"> solution found after "** + str(generationNumber) + **" generations"**)  
 print(**'Time Interval: '** + str(interval) + **' seconds.'**)  
 sys.stdout=StdoutRedirector(textout)  
 print(**"> solution found after "** + str(generationNumber) + **" generations"**)  
 print(**'Time Interval: '** + str(interval) + **' seconds.'**)  
 return population.get\_schedules()[0]  
  
 def handle\_schedule\_display(schedule):  
 while (True):  
 entry=input(  
 **"> What do you want to display (c:lass schedule, t:ime schedule, r:oom schedule, i:nstructor schedule, e:lse)**\n**"**)  
 if (entry == **"c"**):  
 print(**"> from 'class' perspective"**)  
 DisplayMgr.display\_schedule\_as\_table(schedule)  
 elif (entry == **"t"**):  
 print(**"> from 'time' perspective"**)  
 DisplayMgr.display\_schedule\_meetingTimes(schedule)  
 elif (entry == **"r"**):  
 print(**"> from 'room' perspective"**)  
 DisplayMgr.display\_schedule\_rooms(schedule)  
 elif (entry == **"i"**):  
 print(**"> from 'instructor' perspective"**)  
 DisplayMgr.display\_schedule\_instructors(schedule)  
 elif (entry == **"e"**):  
 break  
  
 dbMgr=DBMgr()  
 *# Delete Folders with Trial prefix name* for dirname in glob.glob(**"C:/Program Files/ECE-SPV/Result Saved/Trial \*"**):  
 shutil.rmtree(dirname)  
 ClearSTD()  
 Update()  
 population=Population(POPULATION\_SIZE)  
 start=time.perf\_counter()  
 for m in range(1, NUMBER\_OF\_TRIALS + 1):  
 with open(ARASTDOUT, **'w'**) as f:  
 f.write(**""**)  
 f.close()  
 sys.stdout=open(ARASTDOUT, **'a'**)  
 print(**'**\n**Trial '** + str(m) + **': The algorithm is working for a solution...'**)  
 sys.stdout=StdoutRedirector(textout)  
 print(**'**\n**Trial '** + str(m) + **': The algorithm is working for a solution...'**)  
 dbMgr=DBMgr() *# Refresh Database* schedule=find\_fittest\_schedule(VERBOSE\_FLAG)  
 classes=schedule.get\_classes()  
 conflicts=schedule.get\_conflicts()  
 if (len(conflicts) != 0):  
 conflictList=[]  
 conflictTypeList=[]  
 for i in range(0, len(conflicts)):  
 if (conflicts[i].get\_conflictBetweenClasses()[0] in classes):  
 classes.remove(conflicts[i].get\_conflictBetweenClasses()[0])  
 conflictList.append(conflicts[i].get\_conflictBetweenClasses()[0])  
 conflictTypeList.append(str(conflicts[i].get\_conflictType()))  
 sys.stdout=open(ARASTDOUT, **'a'**)  
 print(**'**\n**Conflict Box (No of Conflicts = '** + str(len(conflicts)) + **')'**)  
 DisplayMgr.display\_schedule\_conflicts(schedule)  
 sys.stdout=StdoutRedirector(textout)  
 print(**'**\n**Conflict Box (No of Conflicts = '** + str(len(conflicts)) + **')'**)  
 DisplayMgr.display\_schedule\_conflicts(schedule)  
  
 instructors=dbMgr.get\_instructors()  
 conflicts=schedule.get\_conflicts()  
 instructorNumbers=[]  
 for i in range(0, len(instructors)):  
 instructorNumbers.append(instructors[i].get\_number())  
  
 overtimeConflicts=[]  
 overloadConflicts=[]  
 instructoravailabilityConflicts=[]  
 roomavailabilityConflicts=[]  
 sectionavailabilityConflicts=[]  
  
 roombookingConflicts=[]  
  
 wrongmthourConflicts=[]  
 overlapConflicts=[]  
 samemtsConflicts=[]  
  
 doubledConflicts=[]  
 *# Segregate Conflicts* for i in range(0, len(conflicts)):  
 if (str(conflicts[i].get\_conflictType()) == **'ConflictType.INSTRUCTOR\_OVERTIME'**):  
 if conflicts[i].get\_conflictBetweenClasses()[0] not in overtimeConflicts:  
 overtimeConflicts.append(conflicts[i].get\_conflictBetweenClasses()[0])  
 elif (str(conflicts[i].get\_conflictType()) == **'ConflictType.INSTRUCTOR\_OVERLOAD'**):  
 if conflicts[i].get\_conflictBetweenClasses()[0] not in overloadConflicts:  
 overloadConflicts.append(conflicts[i].get\_conflictBetweenClasses()[0])  
 elif (str(conflicts[i].get\_conflictType()) == **'ConflictType.INSTRUCTOR\_AVAILABILITY'**):  
 if conflicts[i].get\_conflictBetweenClasses()[0] not in instructoravailabilityConflicts:  
 instructoravailabilityConflicts.append(conflicts[i].get\_conflictBetweenClasses()[0])  
 elif (str(conflicts[i].get\_conflictType()) == **'ConflictType.ROOM\_AVAILABILITY'**):  
 if conflicts[i].get\_conflictBetweenClasses()[0] not in roomavailabilityConflicts:  
 roomavailabilityConflicts.append(conflicts[i].get\_conflictBetweenClasses()[0])  
 elif (str(conflicts[i].get\_conflictType()) == **'ConflictType.SECTION\_AVAILABILITY'**):  
 if conflicts[i].get\_conflictBetweenClasses()[0] not in sectionavailabilityConflicts:  
 sectionavailabilityConflicts.append(conflicts[i].get\_conflictBetweenClasses()[0])  
 elif (str(conflicts[i].get\_conflictType()) == **'ConflictType.WRONG\_MTHOUR'**):  
 if conflicts[i].get\_conflictBetweenClasses()[0] not in wrongmthourConflicts:  
 wrongmthourConflicts.append(conflicts[i].get\_conflictBetweenClasses()[0])  
 elif (str(conflicts[i].get\_conflictType()) == **'ConflictType.OVERLAP\_MTS'**):  
 if conflicts[i].get\_conflictBetweenClasses()[0] not in overlapConflicts:  
 overlapConflicts.append(conflicts[i].get\_conflictBetweenClasses()[0])  
 elif (str(conflicts[i].get\_conflictType()) == **'ConflictType.SAME\_MTS'**):  
 if conflicts[i].get\_conflictBetweenClasses()[0] not in samemtsConflicts:  
 samemtsConflicts.append(conflicts[i].get\_conflictBetweenClasses()[0])  
  
 elif (str(conflicts[i].get\_conflictType()) == **'ConflictType.ROOM\_BOOKING'**):  
 if conflicts[i].get\_conflictBetweenClasses()[0] not in doubledConflicts:  
 doubledConflicts.append(conflicts[i].get\_conflictBetweenClasses()[0])  
 elif (str(conflicts[i].get\_conflictType()) == **'ConflictType.INSTRUCTOR\_BOOKING'**):  
 if conflicts[i].get\_conflictBetweenClasses()[0] not in doubledConflicts:  
 doubledConflicts.append(conflicts[i].get\_conflictBetweenClasses()[0])  
 elif (str(conflicts[i].get\_conflictType()) == **'ConflictType.SECTION\_BOOKING'**):  
 if conflicts[i].get\_conflictBetweenClasses()[0] not in doubledConflicts:  
 doubledConflicts.append(conflicts[i].get\_conflictBetweenClasses()[0])  
 elif (str(conflicts[i].get\_conflictType()) == **'ConflictType.MIXED\_TYPE'**):  
 if conflicts[i].get\_conflictBetweenClasses()[0] not in doubledConflicts:  
 doubledConflicts.append(conflicts[i].get\_conflictBetweenClasses()[0])  
  
  
 *# Removing Conflicts that not suitable for suggestion* for i in range(0, len(doubledConflicts)):  
 if doubledConflicts[i] in overtimeConflicts:  
 overtimeConflicts.remove(doubledConflicts[i])  
 if doubledConflicts[i] in overloadConflicts:  
 overloadConflicts.remove(doubledConflicts[i])  
 for i in range(0, len(overlapConflicts)):  
 if overlapConflicts[i] in overtimeConflicts: *# remove overlapConflicts in overtimeConflicts* overtimeConflicts.remove(overlapConflicts[i])  
 if overlapConflicts[i] in overloadConflicts: *# remove overlapConflicts in overloadConflicts* overloadConflicts.remove(overlapConflicts[i])  
 for i in range(0, len(wrongmthourConflicts)):  
 if wrongmthourConflicts[i] in overtimeConflicts: *# remove wrongmthourConflicts in overtimeConflicts* overtimeConflicts.remove(wrongmthourConflicts[i])  
 if wrongmthourConflicts[i] in overloadConflicts: *# remove wrongmthourConflicts in overloadConflicts* overloadConflicts.remove(wrongmthourConflicts[i])  
 for i in range(0, len(samemtsConflicts)):  
 if samemtsConflicts[i] in overtimeConflicts: *# remove samemtsConflicts in overtimeConflicts* overtimeConflicts.remove(samemtsConflicts[i])  
 if samemtsConflicts[i] in overloadConflicts: *# remove samemtsConflicts in overloadConflicts* overloadConflicts.remove(samemtsConflicts[i])  
 for i in range(0, len(instructoravailabilityConflicts)):  
 if instructoravailabilityConflicts[i] in overtimeConflicts: *# remove instructoravailabilityConflicts in overtimeConflicts* overtimeConflicts.remove(instructoravailabilityConflicts[i])  
 if instructoravailabilityConflicts[i] in overloadConflicts: *# remove instructoravailabilityConflicts in overloadConflicts* overloadConflicts.remove(instructoravailabilityConflicts[i])  
 for i in range(0, len(roomavailabilityConflicts)):  
 if roomavailabilityConflicts[i] in overtimeConflicts: *# remove roomavailabilityConflicts in overtimeConflicts* overtimeConflicts.remove(roomavailabilityConflicts[i])  
 if roomavailabilityConflicts[i] in overloadConflicts: *# remove roomavailabilityConflicts in overloadConflicts* overloadConflicts.remove(roomavailabilityConflicts[i])  
 for i in range(0, len(sectionavailabilityConflicts)):  
 if sectionavailabilityConflicts[i] in overtimeConflicts: *# remove sectionavailabilityConflicts in overtimeConflicts* overtimeConflicts.remove(sectionavailabilityConflicts[i])  
 if sectionavailabilityConflicts[i] in overloadConflicts: *# remove sectionavailabilityConflicts in overloadConflicts* overloadConflicts.remove(sectionavailabilityConflicts[i])  
 for i in range(0, len(doubledConflicts)):  
 if doubledConflicts[i] in overtimeConflicts:  
 overtimeConflicts.remove(doubledConflicts[i])  
 if doubledConflicts[i] in overloadConflicts:  
 overloadConflicts.remove(doubledConflicts[i])  
  
 *# Ensuring the suggested schedules plotted once* for i in range(0, len(overtimeConflicts)):  
 if overtimeConflicts[i] in overloadConflicts: *# remove overtimeConflicts in overloadConflicts* overloadConflicts.remove(overtimeConflicts[i])  
  
 suggestedClasses=overtimeConflicts + overloadConflicts + roombookingConflicts  
 nonsuggestedClasses=[]  
 for i in range(0, len(conflictList)):  
 if conflictList[i] not in suggestedClasses:  
 nonsuggestedClasses.append(conflictList[i])  
 sys.stdout=open(ARASTDOUT, **'a'**)  
 print(**'**\n\n**Total number of Class Schedules that has conflict/s: '** + str(len(conflictList)) + **')'**)  
 print(**'**\n**List of clases that need to schedule manually. (No. of Conflicted Schedule/s: '** + str(  
 len(nonsuggestedClasses)) + **')'**)  
 for i in range(0, len(nonsuggestedClasses)):  
 print(nonsuggestedClasses[i])  
 print(**'**\n**List of Suggested Classes. (No. of Suggestion/s: '** + str(len(suggestedClasses)) + **')'**)  
 for i in range(0, len(suggestedClasses)):  
 print(suggestedClasses[i])  
 sys.stdout=StdoutRedirector(textout)  
 print(**'**\n\n**Total number of Class Schedules that has conflict/s: '** + str(len(conflictList)) + **')'**)  
 print(**'List of classes that need to schedule manually. (No. of Conflicted Schedule/s: '** + str(  
 len(nonsuggestedClasses)) + **')'**)  
 for i in range(0, len(nonsuggestedClasses)):  
 print(nonsuggestedClasses[i])  
 print(**'**\n**List of Suggested Classes. (No. of Suggestion/s: '** + str(len(suggestedClasses)) + **')'**)  
 for i in range(0, len(suggestedClasses)):  
 print(suggestedClasses[i])  
 print(  
 **'**\n**===================================================================================================**\n**'**)  
 print(**'overtimeConflicts'**)  
 for i in range(0, len(overtimeConflicts)):  
 print(overtimeConflicts[i])  
 print(**'overloadConflicts'**)  
 for i in range(0, len(overloadConflicts)):  
 print(overloadConflicts[i])  
 print(**'roombookingConflicts'**)  
 for i in range(0, len(roombookingConflicts)):  
 print(roombookingConflicts[i])  
  
 rooms=dbMgr.get\_rooms()  
 roomNumbers=[]  
 roomConflicts=[]  
 for i in range(0, len(rooms)):  
 roomNumbers.append(rooms[i].get\_number())  
 for i in range(0, len(conflictList)):  
 if (conflictTypeList[i] == **'ConflictType.ROOM\_AVAILABILITY'**):  
 roomConflicts.append(conflictList[i])  
 if (conflictTypeList[i] == **'ConflictType.ROOM\_BOOKING'**):  
 roomConflicts.append(conflictList[i])  
 if (conflictTypeList[i] == **'ConflictType.ROOM\_BOOKING\_OVERLAP'**):  
 roomConflicts.append(conflictList[i])  
 if (conflictTypeList[i] == **'ConflictType.ROOM\_BOOKING\_INBREAK'**):  
 roomConflicts.append(conflictList[i])  
 if (conflictTypeList[i] == **'ConflictType.LAB\_ON\_VR'**):  
 roomConflicts.append(conflictList[i])  
 if (conflictTypeList[i] == **'ConflictType.MIXED\_TYPE'**):  
 roomConflicts.append(conflictList[i])  
 conn=sqlite.connect(**'ARADB.db'**)  
 cursor=conn.cursor()  
 if (len(roomConflicts) >= int(ALLOWABLE\_ROOM\_CONFLICTS\*dbMgr.get\_numberOfClasses())):  
 for n in range(1, 999999999):  
 if (str(**'VR'**) + str(n) not in roomNumbers):  
 cursor.execute(**"INSERT INTO `room` (number, capacity, type, dept) VALUES(?, ?, ?, ?)"**,  
 (str(**'VR'**) + str(n), 100, **'VR'**, **'ECE'**))  
 conn.commit()  
 sys.stdout=open(ARASTDOUT, **'a'**)  
 print(str(**'VR'**) + str(n) + **' is ADDED into Database!**\n**'**)  
 sys.stdout=StdoutRedirector(textout)  
 print(str(**'VR'**) + str(n) + **' is ADDED into Database!**\n**'**)  
 break  
 cursor.close()  
 conn.close()  
 if m == 1:  
 place=**'ST'** elif m == 2:  
 place=**'ND'** elif m == 3:  
 place=**'RD'** else:  
 place=**'TH'** finish=time.perf\_counter()  
 interval=round(finish - start, 2)  
 sys.stdout=open(ARASTDOUT, **'a'**)  
 print(**'**\n**List of successfully scheduled classes.'**)  
 DisplayMgr.display\_schedule\_as\_table(schedule)  
 print(**'The algorithm CANNOT find an optimized solution!'**)  
 print(**'Time Interval: '** + str(interval) + **' seconds.'**)  
 print(**'**\n\n**END OF '** + str(m) + str(place) + **' TRIAL.**\n**'**)  
 sys.stdout=StdoutRedirector(textout)  
 print(**'**\n**List of successfully scheduled classes.'**)  
 DisplayMgr.display\_schedule\_as\_table(schedule)  
 print(**'The algorithm CANNOT find an optimized solution!'**)  
 if (m != NUMBER\_OF\_TRIALS):  
 try:  
 os.system(**'taskkill /F /IM room.xlsx'**)  
 os.system(**'taskkill /F /IM instructor.xlsx'**)  
 os.system(**'taskkill /F /IM section.xlsx'**)  
 os.system(**'taskkill /F /IM xroom.xlsx'**)  
 os.system(**'taskkill /F /IM xinstructor.xlsx'**)  
 os.system(**'taskkill /F /IM xsection.xlsx'**)  
 except OSERROR: pass  
 DisplayMgr.ROOMXLSX(schedule)  
 DisplayMgr.INSTRUCTORXLSX(schedule)  
 DisplayMgr.SECTIONXLSX(schedule)  
 DisplayMgr.XROOMXLSX(schedule)  
 DisplayMgr.XINSTRUCTORXLSX(schedule)  
 DisplayMgr.XSECTIONXLSX(schedule)  
 print(**'Time Interval: '** + str(interval) + **' seconds.'**)  
 print(**'**\n\n**END OF '** + str(m) + str(place) + **' TRIAL.**\n**'**)  
 Trial\_Results()  
 if (m == NUMBER\_OF\_TRIALS):  
 txt\_result.config(text = **"Scheduling results found!"**, fg = **"Green"**)  
 try:  
 os.system(**'taskkill /F /IM room.xlsx'**)  
 os.system(**'taskkill /F /IM instructor.xlsx'**)  
 os.system(**'taskkill /F /IM section.xlsx'**)  
 os.system(**'taskkill /F /IM xroom.xlsx'**)  
 os.system(**'taskkill /F /IM xinstructor.xlsx'**)  
 os.system(**'taskkill /F /IM xsection.xlsx'**)  
 except OSERROR: pass  
 DisplayMgr.ROOMXLSX(schedule)  
 DisplayMgr.INSTRUCTORXLSX(schedule)  
 DisplayMgr.SECTIONXLSX(schedule)  
 DisplayMgr.XROOMXLSX(schedule)  
 DisplayMgr.XINSTRUCTORXLSX(schedule)  
 DisplayMgr.XSECTIONXLSX(schedule)  
 print(**'Time Interval: '** + str(interval) + **' seconds.'**)  
 print(**'**\n\n**END OF '** + str(m) + str(place) + **' TRIAL.**\n**'**)  
 Trial\_Results()  
 Results()  
 os.system(**'start excel.exe room.xlsx'**)  
 os.system(**'start excel.exe instructor.xlsx'**)  
 os.system(**'start excel.exe section.xlsx'**)  
 os.system(**'start excel.exe xroom.xlsx'**)  
 os.system(**'start excel.exe xinstructor.xlsx'**)  
 os.system(**'start excel.exe xsection.xlsx'**)  
 else:  
 if m == 1:  
 place=**'st'** elif m == 2:  
 place=**'nd'** elif m == 3:  
 place=**'rd'** else:  
 place=**'th'** finish=time.perf\_counter()  
 interval=round(finish - start, 2)  
 sys.stdout=open(ARASTDOUT, **'a'**)  
 DisplayMgr.display\_schedule\_as\_table(schedule)  
 print(**'The solution ACQUIRED after '** + str(m) + str(place) + **' trial.**\n\n**'**)  
 print(**'Time Interval: '** + str(interval) + **' seconds.'**)  
 sys.stdout=StdoutRedirector(textout)  
 DisplayMgr.display\_schedule\_as\_table(schedule)  
 print(**'The solution ACQUIRED after '** + str(m) + str(place) + **' trial.**\n\n**'**)  
 print(**'Time Interval: '** + str(interval) + **' seconds.'**)  
 txt\_result.config(text = **"Scheduling results found!"**, fg = **"Green"**)  
 try:  
 os.system(**'taskkill /F /IM room.xlsx'**)  
 os.system(**'taskkill /F /IM section.xlsx'**)  
 os.system(**'taskkill /F /IM instructor.xlsx'**)  
 except OSERROR: pass  
 DisplayMgr.ROOMXLSX(schedule)  
 DisplayMgr.SECTIONXLSX(schedule)  
 DisplayMgr.INSTRUCTORXLSX(schedule)  
 Results()  
 os.system(**'start excel.exe room.xlsx'**)  
 os.system(**'start excel.exe section.xlsx'**)  
 os.system(**'start excel.exe instructor.xlsx'**)  
 break  
 continue  
 print(**'**\n**'**)  
  
  
 def i():  
 txt\_result.config(text = **"View Initial Data Button Clicked!"**, fg = **"#EE7600"**)  
  
 NUMB\_OF\_ELITE\_SCHEDULES=1  
 TOURNAMENT\_SELECTION\_SIZE=3  
  
 class DBMgr:  
 def \_\_init\_\_(self):  
 self.\_conn=sqlite.connect(**'ARADB.db'**) *# sql connector* self.\_cursor=self.\_conn.cursor() *# sql cursor* self.\_times=self.\_select\_times() *# select time sql cursor* self.\_timepoints=self.\_select\_timepoints() *# select timepoints sql cursor* self.\_meetingTimes=self.\_select\_meeting\_times() *# select meeting time sql cursor* self.\_instructors=self.\_select\_instructors() *# select instructor ql cursor* self.\_subjects=self.\_select\_subjects() *# select subject sql cursor* self.\_rooms=self.\_select\_rooms() *# select room sql cursor* self.\_sections=self.\_select\_sections() *# select section sql cursor* self.\_numberOfClasses=0 *# initial number of classes* for i in range(0, len(self.\_sections)):  
 self.\_numberOfClasses+=len(  
 self.\_sections[i].get\_subjects()) *# auto-itereation of number of classes creation  
  
 # select section sql command* def \_select\_sections(self):  
 self.\_cursor.execute(**"SELECT \* FROM section"**)  
 sections=self.\_cursor.fetchall()  
 returnSections=[]  
 for i in range(0, len(sections)):  
 returnSections.append(Section(sections[i][0], self.\_select\_section\_subjects(sections[i][0]),  
 self.\_select\_section\_availability(sections[i][0])))  
 return returnSections  
  
 *# select subject sql command* def \_select\_subjects(self):  
 self.\_cursor.execute(**"SELECT \* FROM subject"**)  
 subjects=self.\_cursor.fetchall()  
 returnSubjects=[]  
 for i in range(0, len(subjects)):  
 returnSubjects.append(  
 Subject(subjects[i][0], subjects[i][1], subjects[i][2], subjects[i][3], subjects[i][4],  
 subjects[i][5], subjects[i][6], subjects[i][7], self.\_select\_subject\_instructors(  
 subjects[i][0])))  
 return returnSubjects  
  
 *# select instructor sql command* def \_select\_instructors(self):  
 self.\_cursor.execute(**"SELECT \* FROM instructor"**)  
 instructors=self.\_cursor.fetchall()  
 returnInstructors=[]  
 for i in range(0, len(instructors)):  
 returnInstructors.append(  
 Instructor(instructors[i][0], instructors[i][1], instructors[i][2], instructors[i][3],  
 self.\_select\_instructor\_availability(instructors[i][0]),  
 self.\_select\_instructor\_availability\_start(instructors[i][0]),  
 self.\_select\_instructor\_availability\_end(instructors[i][0]),  
 self.\_select\_instructor\_overtime(instructors[i][0])))  
 return returnInstructors  
  
 *# select room sql command* def \_select\_rooms(self):  
 self.\_cursor.execute(**"SELECT \* FROM room"**)  
 rooms=self.\_cursor.fetchall()  
 returnRooms=[]  
 for i in range(0, len(rooms)):  
 returnRooms.append(Room(rooms[i][0], rooms[i][1], rooms[i][2], rooms[i][3],  
 self.\_select\_room\_availability(  
 rooms[i][  
 0])))  
 return returnRooms  
  
 *# select meeting time sql command* def \_select\_meeting\_times(self):  
 self.\_cursor.execute(**"SELECT \* FROM meeting\_time"**)  
 meetingTimes=self.\_cursor.fetchall()  
 returnMeetingTimes=[]  
 for i in range(0, len(meetingTimes)):  
 returnMeetingTimes.append(MeetingTime(meetingTimes[i][0], meetingTimes[i][1], meetingTimes[i][2],  
 meetingTimes[i][3], meetingTimes[i][4],  
 self.\_select\_rest\_time(meetingTimes[i][0]),  
 self.\_select\_break\_time(meetingTimes[i][0])))  
 return returnMeetingTimes  
  
 *# select time sql command #3* def \_select\_times(self):  
 self.\_cursor.execute(**"SELECT \* FROM time"**)  
 times=self.\_cursor.fetchall()  
 returnTimes=[]  
 for i in range(0, len(times)):  
 returnTimes.append(Time(times[i][0]))  
 return returnTimes  
  
 *# select timepoint sql command #3* def \_select\_timepoints(self):  
 self.\_cursor.execute(**"SELECT \* FROM time\_point"**)  
 timepoints=self.\_cursor.fetchall()  
 returnTimePoints=[]  
 for i in range(0, len(timepoints)):  
 returnTimePoints.append(TimePoint(timepoints[i][0],  
 self.\_select\_point\_block(timepoints[i][0]),  
 self.\_select\_phantom\_block(timepoints[i][0])))  
 return returnTimePoints  
  
 *# GETTING VALUE* def \_select\_section\_subjects(self, sectionCode):  
 self.\_cursor.execute(**"SELECT \* FROM section\_subject where section\_code == '"** + sectionCode + **"'"**)  
 dbSubjectCodes=self.\_cursor.fetchall()  
 subjectCodes=[]  
 for i in range(0, len(dbSubjectCodes)):  
 subjectCodes.append(dbSubjectCodes[i][1])  
 returnValue=[]  
 for i in range(0, len(self.\_subjects)):  
 if self.\_subjects[i].get\_code() in subjectCodes:  
 returnValue.append(self.\_subjects[i])  
 return returnValue  
  
 def \_select\_subject\_instructors(self, subjectCode):  
 self.\_cursor.execute(**"SELECT \* FROM subject\_instructor where subject\_code == '"** + subjectCode + **"'"**)  
 dbInstructorNumbers=self.\_cursor.fetchall()  
 instructorNumbers=[]  
 for i in range(0, len(dbInstructorNumbers)): instructorNumbers.append(dbInstructorNumbers[i][1])  
 returnValue=[]  
 for i in range(0, len(self.\_instructors)):  
 if self.\_instructors[i].get\_number() in instructorNumbers:  
 returnValue.append(self.\_instructors[i])  
 return returnValue  
  
 def \_select\_instructor\_availability(self, instructor):  
 self.\_cursor.execute(  
 **"SELECT \* from instructor\_availability where instructor\_id == '"** + instructor + **"'"**)  
 instructorMTsRS=self.\_cursor.fetchall()  
 instructorMTs=[]  
 for i in range(0, len(instructorMTsRS)):  
 instructorMTs.append(instructorMTsRS[i][1])  
 instructorAvailability=list()  
 for i in range(0, len(self.\_meetingTimes)):  
 if self.\_meetingTimes[i].get\_id() in instructorMTs:  
 instructorAvailability.append(self.\_meetingTimes[i].get\_id())  
 return instructorAvailability  
  
 def \_select\_instructor\_availability\_start(self, instructor):  
 self.\_cursor.execute(  
 **"SELECT \* from instructor\_availability where instructor\_id == '"** + instructor + **"'"**)  
 instructorMTsRS=self.\_cursor.fetchall()  
 instructorMTsSTART=[]  
 for i in range(0, len(instructorMTsRS)):  
 instructorMTsSTART.append(instructorMTsRS[i][2])  
 instructorAvailabilitySTART=list()  
 for i in range(0, len(self.\_timepoints)):  
 if self.\_timepoints[i].get\_point() in instructorMTsSTART:  
 instructorAvailabilitySTART.append(self.\_timepoints[i].get\_point())  
 return instructorAvailabilitySTART  
  
 def \_select\_instructor\_availability\_end(self, instructor):  
 self.\_cursor.execute(  
 **"SELECT \* from instructor\_availability where instructor\_id == '"** + instructor + **"'"**)  
 instructorMTsRS=self.\_cursor.fetchall()  
 instructorMTsEND=[]  
 for i in range(0, len(instructorMTsRS)):  
 instructorMTsEND.append(instructorMTsRS[i][3])  
 instructorAvailabilityEND=list()  
 for i in range(0, len(self.\_timepoints)):  
 if self.\_timepoints[i].get\_point() in instructorMTsEND:  
 instructorAvailabilityEND.append(self.\_timepoints[i].get\_point())  
 return instructorAvailabilityEND  
  
 def \_select\_instructor\_overtime(self, instructor):  
 self.\_cursor.execute(**"SELECT \* from instructor\_overtime where instructor\_id == '"** + instructor + **"'"**)  
 instructorMTsRS=self.\_cursor.fetchall()  
 instructorMTs=[]  
 for i in range(0, len(instructorMTsRS)):  
 instructorMTs.append(instructorMTsRS[i][1])  
 instructorOvertime=list()  
 for i in range(0, len(self.\_meetingTimes)):  
 if self.\_meetingTimes[i].get\_id() in instructorMTs:  
 instructorOvertime.append(self.\_meetingTimes[i].get\_id())  
 return instructorOvertime  
  
 def \_select\_room\_availability(self, room):  
 self.\_cursor.execute(**"SELECT \* from room\_availability where room\_number = '"** + room + **"'"**)  
 roomMTsRS=self.\_cursor.fetchall()  
 roomMTs=[]  
 for i in range(0, len(roomMTsRS)): roomMTs.append(roomMTsRS[i][1])  
 roomAvailability=list()  
 for i in range(0, len(self.\_meetingTimes)):  
 if self.\_meetingTimes[i].get\_id() in roomMTs:  
 roomAvailability.append(self.\_meetingTimes[i].get\_id())  
 return roomAvailability  
  
 def \_select\_section\_availability(self, section):  
 self.\_cursor.execute(**"SELECT \* from section\_availability where section\_code = '"** + section + **"'"**)  
 sectionMTsRS=self.\_cursor.fetchall()  
 sectionMTs=[]  
 for i in range(0, len(sectionMTsRS)): sectionMTs.append(sectionMTsRS[i][1])  
 sectionAvailability=list()  
 for i in range(0, len(self.\_meetingTimes)):  
 if self.\_meetingTimes[i].get\_id() in sectionMTs:  
 sectionAvailability.append(self.\_meetingTimes[i].get\_id())  
 return sectionAvailability  
  
 def \_select\_rest\_time(self, meeting\_time):  
 self.\_cursor.execute(**"SELECT \* from rest\_time where meeting\_time\_id = '"** + meeting\_time + **"'"**)  
 timeMTsRS=self.\_cursor.fetchall()  
 timeMTs=[]  
 for i in range(0, len(timeMTsRS)): timeMTs.append(timeMTsRS[i][1])  
 restTime=list()  
 for i in range(0, len(self.\_times)):  
 if self.\_times[i].get\_block() in timeMTs:  
 restTime.append(self.\_times[i].get\_block())  
 return restTime  
  
 def \_select\_break\_time(self, meeting\_time):  
 self.\_cursor.execute(**"SELECT \* from break\_time where meeting\_time\_id = '"** + meeting\_time + **"'"**)  
 timeMTsRS=self.\_cursor.fetchall()  
 timeMTs=[]  
 for i in range(0, len(timeMTsRS)): timeMTs.append(timeMTsRS[i][1])  
 breakTime=list()  
 for i in range(0, len(self.\_times)):  
 if self.\_times[i].get\_block() in timeMTs:  
 breakTime.append(self.\_times[i].get\_block())  
 return breakTime  
  
 def \_select\_point\_block(self, point):  
 self.\_cursor.execute(**"SELECT \* from point\_block where time\_point == '"** + point + **"'"**)  
 pointBlocksRS=self.\_cursor.fetchall()  
 pointBlocks=[]  
 for i in range(0, len(pointBlocksRS)): pointBlocks.append(pointBlocksRS[i][1])  
 returnValue=list()  
 for i in range(0, len(self.\_times)):  
 if self.\_times[i].get\_block() in pointBlocks:  
 returnValue.append(self.\_times[i].get\_block())  
 return returnValue  
  
 def \_select\_phantom\_block(self, point):  
 self.\_cursor.execute(**"SELECT \* from phantom\_block where time\_point == '"** + point + **"'"**)  
 pointBlocksRS=self.\_cursor.fetchall()  
 pointBlocks=[]  
 for i in range(0, len(pointBlocksRS)): pointBlocks.append(pointBlocksRS[i][1])  
 returnValue=list()  
 for i in range(0, len(self.\_times)):  
 if self.\_times[i].get\_block() in pointBlocks:  
 returnValue.append(self.\_times[i].get\_block())  
 return returnValue  
  
 def get\_sections(self):  
 return self.\_sections *# get item to section table* def get\_subjects(self):  
 return self.\_subjects *# get item to subject table* def get\_instructors(self):  
 return self.\_instructors *# get item to instructor table* def get\_meetingTimes(self):  
 return self.\_meetingTimes *# get item to meeting time table* def get\_times(self):  
 return self.\_times *# get item to time table* def get\_timepoints(self):  
 return self.\_timepoints *# get item to time table* def get\_rooms(self):  
 return self.\_rooms *# get item to room table* def get\_numberOfClasses(self):  
 return self.\_numberOfClasses *# get item to number of class table* class Section:  
 def \_\_init\_\_(self, code, subjects, availability):  
 self.\_code=code  
 self.\_subjects=subjects  
 self.\_availability=availability  
  
 def get\_code(self): return self.\_code  
  
 def get\_subjects(self): return self.\_subjects  
  
 def get\_availability(self): return self.\_availability  
  
 class Subject:  
 def \_\_init\_\_(self, code, description, units, numbHour, compatibility, curriculum, maxNumbOfStudents, dept,  
 instructors):  
 self.\_code=code  
 self.\_description=description  
 self.\_units=units  
 self.\_numbHour=numbHour  
 self.\_compatibility=compatibility  
 self.\_curriculum=curriculum  
 self.\_maxNumbOfStudents=maxNumbOfStudents  
 self.\_dept=dept  
 self.\_instructors=instructors  
  
 def get\_code(self): return self.\_code  
  
 def get\_description(self): return self.\_description  
  
 def get\_units(self): return self.\_units  
  
 def get\_numbHour(self): return self.\_numbHour  
  
 def get\_compatibility(self): return self.\_compatibility  
  
 def get\_curriculum(self): return self.\_curriculum  
  
 def get\_maxNumbOfStudents(self): return self.\_maxNumbOfStudents  
  
 def get\_dept(self): return self.\_dept  
  
 def get\_instructors(self): return self.\_instructors  
  
 def \_\_str\_\_(self): return self.\_description  
  
 class Time:  
 def \_\_init\_\_(self, block):  
 self.\_block=block  
  
 def get\_block(self): return self.\_block  
  
 def \_\_str\_\_(self): return self.\_block  
  
 class TimePoint:  
 def \_\_init\_\_(self, point, blocks, phantomBlocks):  
 self.\_point=point  
 self.\_blocks=blocks  
 self.\_phantomBlocks=phantomBlocks  
  
 def get\_point(self): return self.\_point  
  
 def get\_blocks(self): return self.\_blocks  
  
 def get\_phantomBlocks(self): return self.\_phantomBlocks  
  
 def \_\_str\_\_(self): return self.\_point  
  
 class Instructor:  
 def \_\_init\_\_(self, number, name, max\_hours, seniority, availability, start, end, overtime):  
 self.\_number=number  
 self.\_name=name  
 self.\_max\_hours=max\_hours  
 self.\_seniority=seniority  
 self.\_availability=availability  
 self.\_start=start  
 self.\_end=end  
 self.\_overtime=overtime  
  
 def get\_number(self): return self.\_number  
  
 def get\_name(self): return self.\_name  
  
 def get\_max\_hours(self): return self.\_max\_hours  
  
 def get\_seniority(self): return self.\_seniority  
  
 def get\_availability(self): return self.\_availability  
  
 def get\_start(self): return self.\_start  
  
 def get\_end(self): return self.\_end  
  
 def get\_overtime(self): return self.\_overtime  
  
 def \_\_str\_\_(self): return self.\_name  
  
 class MeetingTime:  
 def \_\_init\_\_(self, id, time, day, MThour, cell, restTime, breakTime):  
 self.\_id=id  
 self.\_time=time  
 self.\_day=day  
 self.\_MThour=MThour  
 self.\_cell=cell  
 self.\_restTime=restTime   
 self.\_breakTime=breakTime  
  
 def get\_id(self): return self.\_id  
  
 def get\_time(self): return self.\_time  
  
 def get\_day(self): return self.\_day  
  
 def get\_MThour(self): return self.\_MThour  
  
 def get\_cell(self): return self.\_cell  
  
 def get\_restTime(self): return self.\_restTime   
  
 def get\_breakTime(self): return self.\_breakTime  
  
 def \_\_str\_\_(self): return self.\_id  
  
 class MeetingTime1:  
 def \_\_init\_\_(self, id, time, day, MThour, cell, restTime, breakTime):  
 self.\_id=id  
 self.\_time=time  
 self.\_day=day  
 self.\_MThour=MThour  
 self.\_cell=cell  
 self.\_restTime=restTime   
 self.\_breakTime=breakTime  
  
 def get\_id(self): return self.\_id  
  
 def get\_time(self): return self.\_time  
  
 def get\_day(self): return self.\_day  
  
 def get\_MThour(self): return self.\_MThour  
  
 def get\_cell(self): return self.\_cell  
  
 def get\_restTime(self): return self.\_restTime   
  
 def get\_breakTime(self): return self.\_breakTime  
  
 def \_\_str\_\_(self): return self.\_id  
  
 class Room:  
 def \_\_init\_\_(self, number, seatingCapacity, type, dept, availability):  
 self.\_number=number  
 self.\_seatingCapacity=seatingCapacity  
 self.\_type=type  
 self.\_dept=dept  
 self.\_availability=availability  
  
 def get\_number(self): return self.\_number  
  
 def get\_seatingCapacity(self): return self.\_seatingCapacity  
  
 def get\_type(self): return self.\_type  
  
 def get\_dept(self): return self.\_dept  
  
 def get\_availability(self): return self.\_availability  
  
 def \_\_str\_\_(self): return self.\_number  
  
 class DisplayMgr:  
 @staticmethod  
 def display\_input\_data():  
 print(**"> All Available Data"**)  
 DisplayMgr.display\_section()  
 DisplayMgr.display\_subject()  
 DisplayMgr.display\_room()  
 DisplayMgr.display\_instructor()  
  
 @staticmethod  
 def display\_section():  
 sections=dbMgr.get\_sections()  
 availableSectionsTable=prettytable.PrettyTable([**'section'**, **'subjects'**])  
 for i in range(0, len(sections)):  
 subjects=sections.\_\_getitem\_\_(i).get\_subjects()  
 tempStr=**"["** for j in range(0, len(subjects) - 1):  
 tempStr+=subjects[j].\_\_str\_\_() + **", "** tempStr+=subjects[len(subjects) - 1].\_\_str\_\_() + **"]"** availableSectionsTable.add\_row([sections.\_\_getitem\_\_(i).get\_code(), tempStr])  
 print(availableSectionsTable)  
  
 @staticmethod  
 def display\_subject():  
 availableSubjectsTable=prettytable.PrettyTable(  
 [**'ID'**, **'Subject'**, **'Units'**, **'Hours'**, **'Compatibility'**, **'Curriculum'**, **'Max. # of Students'**, **'Dept'**])  
 subjects=dbMgr.get\_subjects()  
 for i in range(0, len(subjects)):  
 availableSubjectsTable.add\_row(  
 [subjects[i].get\_code(), subjects[i].get\_description(), subjects[i].get\_units(),  
 subjects[i].get\_numbHour(),  
 subjects[i].get\_compatibility(), subjects[i].get\_curriculum(),  
 str(subjects[i].get\_maxNumbOfStudents()), str(subjects[i].get\_dept())])   
 print(availableSubjectsTable)  
  
 @staticmethod  
 def display\_instructor():  
 availableInstructorsTable=prettytable.PrettyTable([**'ID'**, **'Instructor'**, **'Max Hours'**, **'Seniority'**])  
 instructors=dbMgr.get\_instructors()  
 for i in range(0, len(instructors)):  
 availableInstructorsTable.add\_row(  
 [instructors[i].get\_number(), instructors[i].get\_name(), instructors[i].get\_max\_hours(),  
 instructors[i].get\_seniority()])  
 print(availableInstructorsTable)  
  
 @staticmethod  
 def display\_room():  
 availableRoomsTable=prettytable.PrettyTable([**'Room'**, **'Max. Capacity'**, **'Type'**, **'Dept'**])  
 rooms=dbMgr.get\_rooms()  
 for i in range(0, len(rooms)):  
 availableRoomsTable.add\_row(  
 [str(rooms[i].get\_number()), str(rooms[i].get\_seatingCapacity()), str(rooms[i].get\_type()),  
 str(rooms[i].get\_dept())])  
 print(availableRoomsTable)  
  
 @staticmethod  
 def display\_meeting\_times():  
 availableMeetingTimeTable=prettytable.PrettyTable([**'ID'**, **'Meeting Time'**, **'Rest Time'**])  
 meetingTimes=dbMgr.get\_meetingTimes()  
 for i in range(0, len(meetingTimes)):  
 restTime=[]  
 for j in range(0, len(meetingTimes[i].get\_restTime())):  
 restTime.append(meetingTimes[i].get\_restTime()[j])  
 availableMeetingTimeTable.add\_row([meetingTimes[i].get\_id(), meetingTimes[i].get\_time(), restTime])  
 print(availableMeetingTimeTable)  
  
 @staticmethod  
 def display\_generation(population):  
 table1=prettytable.PrettyTable(  
 [**'schedule #'**, **'fitness'**, **'# of conflicts'**, **'classes [section,class,room,instructor,meeting-time]'**])  
 schedules=population.get\_schedules()  
 for i in range(0, len(schedules)):  
 table1.add\_row([str(i + 1), round(schedules[i].get\_fitness(), 3), len(schedules[i].get\_conflicts()),  
 schedules[i].\_\_str\_\_()])  
 print(table1)  
  
 @staticmethod  
 def display\_schedule\_as\_table(schedule):  
 classes=schedule.get\_classes()  
 table=prettytable.PrettyTable(  
 [**'Class #'**, **'Section'**, **'Subject (number, max # of students)'**, **'Room (Capacity)'**, **'Instructor (Id)'**,  
 **'Meeting Time 0 (ID)'**, **'Meeting Time 1 (ID)'**])  
 for i in range(0, len(classes)):  
 table.add\_row([str(i + 1), str(classes[i].get\_section().get\_code()),  
 str(classes[i].get\_subject().get\_description()) + **" ("** +  
 str(classes[i].get\_subject().get\_code()) + **", "** +  
 str(classes[i].get\_subject().get\_maxNumbOfStudents()) + **")"**,  
 str(classes[i].get\_room().get\_number()) + **" ("** + str(  
 classes[i].get\_room().get\_seatingCapacity()) + **")"**,  
 str(classes[i].get\_instructor().get\_name()) + **" ("** + str(  
 classes[i].get\_instructor().get\_number()) + **")"**,  
 str(classes[i].get\_meetingTime().get\_time()) + **" ("** + str(  
 classes[i].get\_meetingTime().get\_id()) + **")"**,  
 str(classes[i].get\_meetingTime1().get\_time()) + **" ("** + str(  
 classes[i].get\_meetingTime1().get\_id()) + **")"**])  
 print(table)  
  
 *# Perspectives* @staticmethod  
 def SECTIONXLSX(schedule):  
 sections=dbMgr.get\_sections()  
 sectionCodes=list()  
 section\_instructorBlocks=list()  
 MT\_timeBlocks=list()  
 section\_rooms=list()  
 section\_subjects=list()  
 for i in range(0, len(sections)):  
 sectionCodes.append(sections[i].get\_code())  
 instructorBlock=list()  
 timeBlock=list()  
 roomBlock=list()  
 subjectBlock=list()  
 for j in range(0, len(schedule.get\_classes())):  
 if (schedule.get\_classes()[j].get\_section() == sections[i]):  
 instructorBlock.append(str(schedule.get\_classes()[j].get\_instructor().get\_number()))  
 timeBlock.append(str(schedule.get\_classes()[j].get\_meetingTime().get\_id()))  
 roomBlock.append(str(schedule.get\_classes()[j].get\_room().get\_number()))  
 subjectBlock.append(str(schedule.get\_classes()[j].get\_subject().get\_code()))  
 for j in range(0, len(schedule.get\_classes())):  
 if (schedule.get\_classes()[j].get\_subject().get\_numbHour() > schedule.get\_classes()[  
 j].get\_meetingTime().get\_MThour()):  
 if (schedule.get\_classes()[j].get\_section() == sections[i]):  
 instructorBlock.append(str(schedule.get\_classes()[j].get\_instructor().get\_number()))  
 timeBlock.append(str(schedule.get\_classes()[j].get\_meetingTime1().get\_id()))  
 roomBlock.append(str(schedule.get\_classes()[j].get\_room().get\_number()))  
 subjectBlock.append(str(schedule.get\_classes()[j].get\_subject().get\_code()))  
 section\_instructorBlocks.append(instructorBlock)  
 MT\_timeBlocks.append(timeBlock)  
 section\_rooms.append(roomBlock)  
 section\_subjects.append(subjectBlock)  
 timeCells=dbMgr.get\_meetingTimes()  
 meeetingTimes=list()  
 cells=list()  
 for i in range(0, len(timeCells)):  
 meeetingTimes.append(timeCells[i].get\_id())  
 cells.append(timeCells[i].get\_cell())  
  
 *# create file (workbook) and worksheet* outWorkbook=xlsxwriter.Workbook(**"section.xlsx"**)  
  
 def SECTIONBLOCKS():  
 *# declare data* columnData=[**"Time"**, **"Monday"**, **"Tuesday"**, **"Wednesday"**, **"Thursday"**, **"Friday"**, **"Saturday"**, **"Sunday"**]  
 timeBlock=[**"7:00 - 7:30"**, **"7:30 - 8:00"**, **"8:00 - 8:30"**, **"8:30 - 9:00"**, **"9:00 - 9:30"**,  
 **"9:30 - 10:00"**,  
 **"10:00 - 10:30"**,  
 **"10:30 - 11:00"**, **"11:00 - 11:30"**, **"11:30 - 12:00"**, **"12:00 - 12:30"**, **"12:30 - 1:00"**,  
 **"1:00 - 1:30"**,  
 **"1:30 -2:00"**, **"2:00 - 2:30"**, **"2:30 - 3:00"**, **"3:00 - 3:30"**, **"3:30 - 4:00"**,  
 **"4:00 - 4:30"**,  
 **"4:30 - 5:00"**,  
 **"5:00 - 5:30"**, **"5:30 - 6:00"**, **"6:00 - 6:30"**, **"6:30 - 7:00"**, **"7:00 - 7:30"**,  
 **"7:30 - 8:00"**,  
 **"8:00 - 8:30"**,  
 **"8:30 - 9:00"**]  
 load\_title=**'STUDENT SCHEDULE'** summaryColumn=[**"SUBJECT"**, **"TIME"**, **"DAY"**, **"ROOM"**, **"UNIT"**, **"# OF STUDENTS"**, **"FACULTY"**]  
 outputSheet=outWorkbook.add\_worksheet(name = sectionCodes[sheet])  
 *# Increase the cell size of the merged cells to highlight the formatting.* outputSheet.set\_column(**'A:H'**, 15)  
 outputSheet.set\_default\_row(20)  
 *# Create a format to use in the...  
 # ...centerBoldHead.* centerBoldHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...centerBoldHeadless.* centerBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...leftBoldHeadless.* leftBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...uleftHeadless.* uleftHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...rightBoldHeadless.* rightBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 0,  
 **'align'**: **'right'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...centerHead.* centerHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...ucenterHeadless.* ucenterHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...icenterHeadless.* icenterHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'italic'**: 1,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...smallcenterHead.* smallcenterHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 7.5,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...xsmallcenterHead.* xsmallcenterHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 7.5,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'fadfca'**})  
 *# ...ysmallcenterHead.* ysmallcenterHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 7.5,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'ffe338'**})  
 *# ...leftHeadless.* leftHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...details.* sheet\_details=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vbottom'**,  
 **'fg\_color'**: **'white'**})  
 *# ...header.* header=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 22,  
 **'font\_color'**: **'white'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'c31d3a'**}) *# Green, Accent 6, Darker 25%: 76933C  
 # ... columns.* column=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'white'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'c41e39'**}) *# Green, Accent 6: 9BBB59  
 # ...time.* time=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**}) *# Green, Accent 6, Lighter 40%: C4D798  
 # ...classSched.* classSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'ececec'**}) *# Emerald: 50C878  
 # Write inside the Cell  
 # ...columnData* for item in range(len(columnData)):  
 outputSheet.write(11, item + 0, columnData[item], column)  
 *# ...timeBlock* for item in range(len(timeBlock)):  
 outputSheet.write(item + 12, 0, timeBlock[item], time)  
 *# ...Merge School* outputSheet.merge\_range(**'B1:F4'**, **''**, centerHead)  
 outputSheet.write\_rich\_string(**'B1'**, centerBoldHead, schooltext, centerHead, school\_infotext,  
 centerHead)  
 *# ...Load Title* outputSheet.merge\_range(**'B5:F6'**, load\_title, centerBoldHead)  
 *# ...VAA-OAP* outputSheet.merge\_range(**'A5:A6'**, vaa\_oap, centerHead)  
 *# ...Sheet Details* outputSheet.write\_string(**'G1'**, **'Index No.'**, sheet\_details)  
 outputSheet.write\_string(**'G2'**, **'Issue No.'**, sheet\_details)  
 outputSheet.write\_string(**'G3'**, **'Revision No.'**, sheet\_details)  
 outputSheet.write\_string(**'G4'**, **'Date'**, sheet\_details)  
 outputSheet.write\_string(**'G5'**, **'Page'**, sheet\_details)  
 outputSheet.write\_string(**'G6'**, **'QAC No.'**, sheet\_details)  
 outputSheet.write\_string(**'H1'**, str(indexno), sheet\_details)  
 outputSheet.write\_string(**'H2'**, str(issueno), sheet\_details)  
 outputSheet.write\_string(**'H3'**, str(revisionno), sheet\_details)  
 outputSheet.write\_string(**'H4'**, str(date), sheet\_details)  
 outputSheet.write\_string(**'H5'**, str(page), sheet\_details)  
 outputSheet.write\_string(**'H6'**, str(qacno), sheet\_details)  
 *# ...Logo* outputSheet.merge\_range(**'A1:A4'**, **''**, centerHead)  
 outputSheet.insert\_image(**'A1'**, **'tup-logo.png'**, {**'x\_scale'**: 1.1, **'y\_scale'**: 1.1})  
 *# ...College* outputSheet.merge\_range(**'A8:B8'**, **' COLLEGE:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C8:E8'**,  
 **'\_'** + college + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...Department* outputSheet.merge\_range(**'A9:B9'**, **' DEPARTMENT:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C9:E9'**,  
 **'\_'** + department + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...Name of Faculty* outputSheet.merge\_range(**'A10:B10'**, **' SECTION:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C10:E10'**,  
 **'\_'** + str(  
 sectionCodes[sheet]) + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...S.Y.* outputSheet.write\_string(**'G8'**, **'S.Y.:'**, rightBoldHeadless)  
 outputSheet.write\_string(**'H8'**, **'\_\_'** + schoolyear + **'\_\_'**, uleftHeadless)  
 *# ...Semester* outputSheet.write\_string(**'G9'**, **'Semester:'**, rightBoldHeadless)  
 outputSheet.write\_string(**'H9'**, **'\_'** + semester + **'\_'**, uleftHeadless)  
 *# ...bordercells* for row in range(12, 40):  
 for col in range(1, 8):  
 outputSheet.write(row, col, **''**, centerHead)  
 *# Summary Table  
 # ...summaryColumn* for item in range(len(summaryColumn)):  
 outputSheet.write(43, item, summaryColumn[item], column)  
 outputSheet.merge\_range(43, 6, 43, 7, summaryColumn[6], column)  
  
 for row in range(44, 56):  
 for col in range(0, 6):  
 outputSheet.write(row, col, **''**, centerHead)  
 outputSheet.merge\_range(row, 6, row, 7, **''**, centerHead)  
 classes=schedule.get\_classes()  
 class\_list=list()  
 for i in range(0, len(classes)):  
 if (classes[i].get\_section().get\_code() == sectionCodes[sheet]):  
 class\_list.append(classes[i])  
 for i in range(0, len(class\_list)):  
 if class\_list[i].get\_section().get\_code() == sectionCodes[sheet]:  
 outputSheet.write(44 + i, 0, class\_list[i].get\_subject().get\_code(), smallcenterHead)  
 if class\_list[i].get\_meetingTime1().get\_id() != **'NULL'**:  
 outputSheet.write(44 + i, 1,  
 str(class\_list[i].get\_meetingTime().get\_time()) + **'**\n**'** + str(  
 class\_list[i].get\_meetingTime1().get\_time()), smallcenterHead)  
 outputSheet.write(44 + i, 2,  
 str(class\_list[i].get\_meetingTime().get\_day()[0:3]) + **'**\n**'** + str(  
 class\_list[i].get\_meetingTime1().get\_day()[0:3]), smallcenterHead)  
 else:  
 outputSheet.write(44 + i, 1, class\_list[i].get\_meetingTime().get\_time(),  
 smallcenterHead)  
 outputSheet.write(44 + i, 2, class\_list[i].get\_meetingTime().get\_day()[0:3],  
 smallcenterHead)  
 outputSheet.write(44 + i, 3, class\_list[i].get\_room().get\_number(), smallcenterHead)  
 outputSheet.write(44 + i, 4, class\_list[i].get\_subject().get\_units(), smallcenterHead)  
 outputSheet.write(44 + i, 5, class\_list[i].get\_subject().get\_maxNumbOfStudents(),  
 smallcenterHead)  
 outputSheet.write(44 + i, 6, class\_list[i].get\_instructor().get\_name(), smallcenterHead)  
  
 *# ...Authority Sign* outputSheet.merge\_range(**'A57:C57'**, **' Prepared By:'**, leftHeadless)  
 outputSheet.merge\_range(**'A59:C59'**, **'\_\_\_\_\_'** + str(depthead) + **'\_\_\_\_\_'**, ucenterHeadless)  
 outputSheet.merge\_range(**'A60:C60'**, **'Department Head'**, icenterHeadless)  
 outputSheet.merge\_range(**'F57:H57'**, **' Recommending Approval:'**, leftHeadless)  
 outputSheet.merge\_range(**'F59:H59'**, **'\_\_\_\_\_'** + str(dean) + **'\_\_\_\_\_'**, ucenterHeadless)  
 outputSheet.merge\_range(**'F60:H60'**, **'College Dean'**, icenterHeadless)  
 *# ...footer* outputSheet.write\_string(**'B62'**, **'Transaction ID'**, sheet\_details)  
 outputSheet.merge\_range(**'C62:G62'**, **''**, sheet\_details)  
 outputSheet.write\_string(**'B63'**, **'Signature'**, sheet\_details)  
 outputSheet.merge\_range(**'C63:G63'**, **''**, sheet\_details)  
  
 for MT in MT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in MT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 section\_instructorBlocks[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 section\_rooms[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 section\_subjects[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 MT\_timeBlocks[sheet][MT\_timeBlocks[sheet].index(MT)]), classSched)  
  
 for numb\_of\_sections in range(0, len(sections)):  
 sheet=numb\_of\_sections  
 SECTIONBLOCKS()  
 outWorkbook.close()  
  
 @staticmethod  
 def ROOMXLSX(schedule):  
 rooms=dbMgr.get\_rooms()  
 roomNumbers=list()  
 room\_sectionBlocks=list()  
 MT\_timeBlocks=list()  
 room\_instructors=list()  
 room\_subjects=list()  
 for i in range(0, len(rooms)):  
 roomNumbers.append(rooms[i].get\_number())  
 sectionBlock=list()  
 instructorBlock=list()  
 subjectBlock=list()  
 timeBlock=list()  
 for j in range(0, len(schedule.get\_classes())):  
 if (schedule.get\_classes()[j].get\_room() == rooms[i]):  
 sectionBlock.append(str(schedule.get\_classes()[j].get\_section().get\_code()))  
 instructorBlock.append(str(schedule.get\_classes()[j].get\_instructor().get\_number()))  
 subjectBlock.append(str(schedule.get\_classes()[j].get\_subject().get\_code()))  
 timeBlock.append(str(schedule.get\_classes()[j].get\_meetingTime().get\_id()))  
 for j in range(0, len(schedule.get\_classes())):  
 if (schedule.get\_classes()[j].get\_subject().get\_numbHour() > schedule.get\_classes()[  
 j].get\_meetingTime().get\_MThour()):  
 if (schedule.get\_classes()[j].get\_room() == rooms[i]):  
 sectionBlock.append(str(schedule.get\_classes()[j].get\_section().get\_code()))  
 instructorBlock.append(str(schedule.get\_classes()[j].get\_instructor().get\_number()))  
 subjectBlock.append(str(schedule.get\_classes()[j].get\_subject().get\_code()))  
 timeBlock.append(str(schedule.get\_classes()[j].get\_meetingTime1().get\_id()))  
 room\_sectionBlocks.append(sectionBlock)  
 room\_instructors.append(instructorBlock)  
 room\_subjects.append(subjectBlock)  
 MT\_timeBlocks.append(timeBlock)  
 timeCells=dbMgr.get\_meetingTimes()  
 meeetingTimes=list()  
 cells=list()  
 for i in range(0, len(timeCells)):  
 meeetingTimes.append(timeCells[i].get\_id())  
 cells.append(timeCells[i].get\_cell())  
  
 *# create file (workbook) and worksheet* outWorkbook=xlsxwriter.Workbook(**"room.xlsx"**)  
  
 def ROOMBLOCKS():  
 *# declare data* columnData=[**"Time"**, **"Monday"**, **"Tuesday"**, **"Wednesday"**, **"Thursday"**, **"Friday"**, **"Saturday"**, **"Sunday"**]  
 timeBlock=[**"7:00 - 7:30"**, **"7:30 - 8:00"**, **"8:00 - 8:30"**, **"8:30 - 9:00"**, **"9:00 - 9:30"**,  
 **"9:30 - 10:00"**,  
 **"10:00 - 10:30"**,  
 **"10:30 - 11:00"**, **"11:00 - 11:30"**, **"11:30 - 12:00"**, **"12:00 - 12:30"**, **"12:30 - 1:00"**,  
 **"1:00 - 1:30"**,  
 **"1:30 -2:00"**, **"2:00 - 2:30"**, **"2:30 - 3:00"**, **"3:00 - 3:30"**, **"3:30 - 4:00"**,  
 **"4:00 - 4:30"**,  
 **"4:30 - 5:00"**,  
 **"5:00 - 5:30"**, **"5:30 - 6:00"**, **"6:00 - 6:30"**, **"6:30 - 7:00"**, **"7:00 - 7:30"**,  
 **"7:30 - 8:00"**,  
 **"8:00 - 8:30"**,  
 **"8:30 - 9:00"**]  
 load\_title=**'ROOM SCHEDULE'** outputSheet=outWorkbook.add\_worksheet(name = roomNumbers[sheet])  
 *# Increase the cell size of the merged cells to highlight the formatting.* outputSheet.set\_column(**'A:H'**, 15)  
 outputSheet.set\_default\_row(20)  
 *# Create a format to use in the...  
 # ...centerBoldHead.* centerBoldHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...centerBoldHeadless.* centerBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...leftBoldHeadless.* leftBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...uleftHeadless.* uleftHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...rightBoldHeadless.* rightBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 0,  
 **'align'**: **'right'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...centerHead.* centerHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...ucenterHeadless.* ucenterHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...icenterHeadless.* icenterHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'italic'**: 1,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...smallcenterHead.* smallcenterHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 7.5,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...leftHeadless.* leftHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...details.* sheet\_details=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vbottom'**,  
 **'fg\_color'**: **'white'**})  
 *# ...header.* header=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 22,  
 **'font\_color'**: **'white'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'c31d3a'**}) *# Green, Accent 6, Darker 25%: 76933C  
 # ... columns.* column=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'white'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'c41e39'**}) *# Green, Accent 6: 9BBB59  
 # ...time.* time=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**}) *# Green, Accent 6, Lighter 40%: C4D798  
 # ...classSched.* classSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'ececec'**}) *# Emerald: 50C878  
 # Write inside the Cell  
 # ...columnData* for item in range(len(columnData)):  
 outputSheet.write(11, item + 0, columnData[item], column)  
 *# ...timeBlock* for item in range(len(timeBlock)):  
 outputSheet.write(item + 12, 0, timeBlock[item], time)  
 *# ...Merge School* outputSheet.merge\_range(**'B1:F4'**, **''**, centerHead)  
 outputSheet.write\_rich\_string(**'B1'**, centerBoldHead, schooltext, centerHead, school\_infotext,  
 centerHead)  
 *# ...Load Title* outputSheet.merge\_range(**'B5:F6'**, load\_title, centerBoldHead)  
 *# ...VAA-OAP* outputSheet.merge\_range(**'A5:A6'**, vaa\_oap, centerHead)  
 *# ...Sheet Details* outputSheet.write\_string(**'G1'**, **'Index No.'**, sheet\_details)  
 outputSheet.write\_string(**'G2'**, **'Issue No.'**, sheet\_details)  
 outputSheet.write\_string(**'G3'**, **'Revision No.'**, sheet\_details)  
 outputSheet.write\_string(**'G4'**, **'Date'**, sheet\_details)  
 outputSheet.write\_string(**'G5'**, **'Page'**, sheet\_details)  
 outputSheet.write\_string(**'G6'**, **'QAC No.'**, sheet\_details)  
 outputSheet.write\_string(**'H1'**, str(indexno), sheet\_details)  
 outputSheet.write\_string(**'H2'**, str(issueno), sheet\_details)  
 outputSheet.write\_string(**'H3'**, str(revisionno), sheet\_details)  
 outputSheet.write\_string(**'H4'**, str(date), sheet\_details)  
 outputSheet.write\_string(**'H5'**, str(page), sheet\_details)  
 outputSheet.write\_string(**'H6'**, str(qacno), sheet\_details)  
 *# ...Logo* outputSheet.merge\_range(**'A1:A4'**, **''**, centerHead)  
 outputSheet.insert\_image(**'A1'**, **'tup-logo.png'**, {**'x\_scale'**: 1.1, **'y\_scale'**: 1.1})  
 *# ...College* outputSheet.merge\_range(**'A8:B8'**, **' COLLEGE:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C8:E8'**,  
 **'\_'** + college + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...Department* outputSheet.merge\_range(**'A9:B9'**, **' DEPARTMENT:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C9:E9'**,  
 **'\_'** + department + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...Name of Faculty* outputSheet.merge\_range(**'A10:B10'**, **' ROOM:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C10:E10'**,  
 **'\_'** + str(  
 roomNumbers[sheet]) + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...S.Y.* outputSheet.write\_string(**'G8'**, **'S.Y.:'**, rightBoldHeadless)  
 outputSheet.write\_string(**'H8'**, **'\_\_'** + schoolyear + **'\_\_'**, uleftHeadless)  
 *# ...Semester* outputSheet.write\_string(**'G9'**, **'Semester:'**, rightBoldHeadless)  
 outputSheet.write\_string(**'H9'**, **'\_'** + semester + **'\_'**, uleftHeadless)  
 *# ...bordercells* for row in range(12, 40):  
 for col in range(1, 8):  
 outputSheet.write(row, col, **''**, centerHead)  
 *# ...Authority Sign* outputSheet.merge\_range(**'A44:C44'**, **' Prepared By:'**, leftHeadless)  
 outputSheet.merge\_range(**'A46:C46'**, **'\_\_\_\_\_'** + str(depthead) + **'\_\_\_\_\_'**, ucenterHeadless)  
 outputSheet.merge\_range(**'A47:C47'**, **'Department Head'**, icenterHeadless)  
 outputSheet.merge\_range(**'F44:H44'**, **' Recommending Approval:'**, leftHeadless)  
 outputSheet.merge\_range(**'F46:H46'**, **'\_\_\_\_\_'** + str(dean) + **'\_\_\_\_\_'**, ucenterHeadless)  
 outputSheet.merge\_range(**'F47:H47'**, **'College Dean'**, icenterHeadless)  
 *# ...footer* outputSheet.write\_string(**'B50'**, **'Transaction ID'**, sheet\_details)  
 outputSheet.merge\_range(**'C50:G50'**, **''**, sheet\_details)  
 outputSheet.write\_string(**'B51'**, **'Signature'**, sheet\_details)  
 outputSheet.merge\_range(**'C51:G51'**, **''**, sheet\_details)  
  
 for MT in MT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in MT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 room\_instructors[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 room\_sectionBlocks[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 room\_subjects[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 MT\_timeBlocks[sheet][MT\_timeBlocks[sheet].index(MT)]), classSched)  
  
 for numb\_of\_rooms in range(0, len(rooms)):  
 sheet=numb\_of\_rooms  
 ROOMBLOCKS()  
 outWorkbook.close()  
  
 @staticmethod  
 def INSTRUCTORXLSX(schedule):  
 instructors=dbMgr.get\_instructors()  
 instructorIDs=list()  
 instructor\_sectionBlocks=list()  
 MT\_timeBlocks=list()  
 instructor\_rooms=list()  
 instructor\_subjects=list()  
 for i in range(0, len(instructors)):  
 instructorIDs.append(instructors[i].get\_name())  
 sectionBlock=list()  
 timeBlock=list()  
 roomBlock=list()  
 subjectBlock=list()  
 for j in range(0, len(schedule.get\_classes())):  
 if (schedule.get\_classes()[j].get\_instructor() == instructors[i]):  
 sectionBlock.append(str(schedule.get\_classes()[j].get\_section().get\_code()))  
 timeBlock.append(str(schedule.get\_classes()[j].get\_meetingTime().get\_id()))  
 roomBlock.append(str(schedule.get\_classes()[j].get\_room().get\_number()))  
 subjectBlock.append(str(schedule.get\_classes()[j].get\_subject().get\_code()))  
 for j in range(0, len(schedule.get\_classes())):  
 if (schedule.get\_classes()[j].get\_subject().get\_numbHour() > schedule.get\_classes()[  
 j].get\_meetingTime().get\_MThour()):  
 if (schedule.get\_classes()[j].get\_instructor() == instructors[i]):  
 sectionBlock.append(str(schedule.get\_classes()[j].get\_section().get\_code()))  
 timeBlock.append(str(schedule.get\_classes()[j].get\_meetingTime1().get\_id()))  
 roomBlock.append(str(schedule.get\_classes()[j].get\_room().get\_number()))  
 subjectBlock.append(str(schedule.get\_classes()[j].get\_subject().get\_code()))  
 instructor\_sectionBlocks.append(sectionBlock)  
 MT\_timeBlocks.append(timeBlock)  
 instructor\_rooms.append(roomBlock)  
 instructor\_subjects.append(subjectBlock)  
 timeCells=dbMgr.get\_meetingTimes()  
 meeetingTimes=list()  
 cells=list()  
 for i in range(0, len(timeCells)):  
 meeetingTimes.append(timeCells[i].get\_id())  
 cells.append(timeCells[i].get\_cell())  
  
 *# create file (workbook) and worksheet* outWorkbook=xlsxwriter.Workbook(**"instructor.xlsx"**)  
  
 def INSTRUCTORBLOCKS():  
 *# declare data* columnData=[**"Time"**, **"Monday"**, **"Tuesday"**, **"Wednesday"**, **"Thursday"**, **"Friday"**, **"Saturday"**, **"Sunday"**]  
 timeBlock=[**"7:00 - 7:30"**, **"7:30 - 8:00"**, **"8:00 - 8:30"**, **"8:30 - 9:00"**, **"9:00 - 9:30"**,  
 **"9:30 - 10:00"**,  
 **"10:00 - 10:30"**,  
 **"10:30 - 11:00"**, **"11:00 - 11:30"**, **"11:30 - 12:00"**, **"12:00 - 12:30"**, **"12:30 - 1:00"**,  
 **"1:00 - 1:30"**,  
 **"1:30 -2:00"**, **"2:00 - 2:30"**, **"2:30 - 3:00"**, **"3:00 - 3:30"**, **"3:30 - 4:00"**,  
 **"4:00 - 4:30"**,  
 **"4:30 - 5:00"**,  
 **"5:00 - 5:30"**, **"5:30 - 6:00"**, **"6:00 - 6:30"**, **"6:30 - 7:00"**, **"7:00 - 7:30"**,  
 **"7:30 - 8:00"**,  
 **"8:00 - 8:30"**,  
 **"8:30 - 9:00"**]  
 load\_title=**'INDIVIDUAL FACULTY LOADING'** outputSheet=outWorkbook.add\_worksheet(name = instructorIDs[sheet])  
 *# Increase the cell size of the merged cells to highlight the formatting.* outputSheet.set\_column(**'A:H'**, 15)  
 outputSheet.set\_default\_row(20)  
 *# Create a format to use in the...  
 # ...centerBoldHead.* centerBoldHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...centerBoldHeadless.* centerBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...leftBoldHeadless.* leftBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...uleftHeadless.* uleftHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...rightBoldHeadless.* rightBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 0,  
 **'align'**: **'right'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...centerHead.* centerHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...ucenterHeadless.* ucenterHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...icenterHeadless.* icenterHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'italic'**: 1,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...smallcenterHead.* smallcenterHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 7.5,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...leftHeadless.* leftHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...details.* sheet\_details=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vbottom'**,  
 **'fg\_color'**: **'white'**})  
 *# ...header.* header=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 22,  
 **'font\_color'**: **'white'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'c31d3a'**}) *# Green, Accent 6, Darker 25%: 76933C  
 # ... columns.* column=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'white'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'c41e39'**}) *# Green, Accent 6: 9BBB59  
 # ...time.* time=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**}) *# Green, Accent 6, Lighter 40%: C4D798  
 # ...classSched.* classSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'ececec'**}) *# Emerald: 50C878  
 # Write inside the Cell  
 # ...columnData* for item in range(len(columnData)):  
 outputSheet.write(11, item + 0, columnData[item], column)  
 *# ...timeBlock* for item in range(len(timeBlock)):  
 outputSheet.write(item + 12, 0, timeBlock[item], time)  
 *# ...Merge School* outputSheet.merge\_range(**'B1:F4'**, **''**, centerHead)  
 outputSheet.write\_rich\_string(**'B1'**, centerBoldHead, schooltext, centerHead, school\_infotext,  
 centerHead)  
 *# ...Load Title* outputSheet.merge\_range(**'B5:F6'**, load\_title, centerBoldHead)  
 *# ...VAA-OAP* outputSheet.merge\_range(**'A5:A6'**, vaa\_oap, centerHead)  
 *# ...Sheet Details* outputSheet.write\_string(**'G1'**, **'Index No.'**, sheet\_details)  
 outputSheet.write\_string(**'G2'**, **'Issue No.'**, sheet\_details)  
 outputSheet.write\_string(**'G3'**, **'Revision No.'**, sheet\_details)  
 outputSheet.write\_string(**'G4'**, **'Date'**, sheet\_details)  
 outputSheet.write\_string(**'G5'**, **'Page'**, sheet\_details)  
 outputSheet.write\_string(**'G6'**, **'QAC No.'**, sheet\_details)  
 outputSheet.write\_string(**'H1'**, str(indexno), sheet\_details)  
 outputSheet.write\_string(**'H2'**, str(issueno), sheet\_details)  
 outputSheet.write\_string(**'H3'**, str(revisionno), sheet\_details)  
 outputSheet.write\_string(**'H4'**, str(date), sheet\_details)  
 outputSheet.write\_string(**'H5'**, str(page), sheet\_details)  
 outputSheet.write\_string(**'H6'**, str(qacno), sheet\_details)  
 *# ...Logo* outputSheet.merge\_range(**'A1:A4'**, **''**, centerHead)  
 outputSheet.insert\_image(**'A1'**, **'tup-logo.png'**, {**'x\_scale'**: 1.1, **'y\_scale'**: 1.1})  
 *# ...College* outputSheet.merge\_range(**'A8:B8'**, **' COLLEGE:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C8:E8'**,  
 **'\_'** + college + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...Department* outputSheet.merge\_range(**'A9:B9'**, **' DEPARTMENT:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C9:E9'**,  
 **'\_'** + department + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...Name of Faculty* outputSheet.merge\_range(**'A10:B10'**, **' NAME OF FACULTY:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C10:E10'**,  
 **'\_'** + str(  
 instructorIDs[sheet]) + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...S.Y.* outputSheet.write\_string(**'G8'**, **'S.Y.:'**, rightBoldHeadless)  
 outputSheet.write\_string(**'H8'**, **'\_\_'** + schoolyear + **'\_\_'**, uleftHeadless)  
 *# ...Semester* outputSheet.write\_string(**'G9'**, **'Semester:'**, rightBoldHeadless)  
 outputSheet.write\_string(**'H9'**, **'\_'** + semester + **'\_'**, uleftHeadless)  
 *# ...bordercells* for row in range(12, 40):  
 for col in range(1, 8):  
 outputSheet.write(row, col, **''**, centerHead)  
 *# ...below table* outputSheet.merge\_range(**'A44:B44'**, **' OFFICIAL TIME:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'A47:B47'**, **' OVERLOAD:'**, leftBoldHeadless)  
 outputSheet.write\_string(**'C43'**, **'DAY'**, centerBoldHeadless)  
 outputSheet.write\_string(**'C44'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.write\_string(**'C45'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.write\_string(**'C46'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.write\_string(**'C47'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.write\_string(**'C48'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.write\_string(**'C49'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.merge\_range(**'E43:F43'**, **'TIME'**, centerBoldHeadless)  
 outputSheet.merge\_range(**'E44:F44'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.merge\_range(**'E45:F45'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.merge\_range(**'E46:F46'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.merge\_range(**'E47:F47'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.merge\_range(**'E48:F48'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.merge\_range(**'E49:F49'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 *# ...footer* outputSheet.write\_string(**'B50'**, **'Transaction ID'**, sheet\_details)  
 outputSheet.merge\_range(**'C50:G50'**, **''**, sheet\_details)  
 outputSheet.write\_string(**'B51'**, **'Signature'**, sheet\_details)  
 outputSheet.merge\_range(**'C51:G51'**, **''**, sheet\_details)  
  
 for MT in MT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in MT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 instructor\_sectionBlocks[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 instructor\_subjects[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 instructor\_rooms[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 MT\_timeBlocks[sheet][MT\_timeBlocks[sheet].index(MT)]), classSched)  
  
 for numb\_of\_instructors in range(0, len(instructors)):  
 sheet=numb\_of\_instructors  
 INSTRUCTORBLOCKS()  
 outWorkbook.close()  
  
 @staticmethod  
 def XSECTIONXLSX(schedule):  
 sections=dbMgr.get\_sections()  
 sectionCodes=list()  
 section\_instructorBlocks=list()  
 MT\_timeBlocks=list()  
 section\_rooms=list()  
 section\_subjects=list()  
 xsection\_instructorBlocks=list()  
 xMT\_timeBlocks=list()  
 xsection\_rooms=list()  
 xsection\_subjects=list()  
 ysection\_instructorBlocks=list()  
 yMT\_timeBlocks=list()  
 ysection\_rooms=list()  
 ysection\_subjects=list()  
 for i in range(0, len(sections)):  
 sectionCodes.append(sections[i].get\_code())  
 instructorBlock=list()  
 timeBlock=list()  
 roomBlock=list()  
 subjectBlock=list()  
 xinstructorBlock=list()  
 xtimeBlock=list()  
 xroomBlock=list()  
 xsubjectBlock=list()  
 yinstructorBlock=list()  
 ytimeBlock=list()  
 yroomBlock=list()  
 ysubjectBlock=list()  
 for j in range(0, len(schedule.get\_classes())):  
 if (schedule.get\_classes()[j].get\_section() == sections[i]):  
 instructorBlock.append(str(schedule.get\_classes()[j].get\_instructor().get\_number()))  
 timeBlock.append(str(schedule.get\_classes()[j].get\_meetingTime().get\_id()))  
 roomBlock.append(str(schedule.get\_classes()[j].get\_room().get\_number()))  
 subjectBlock.append(str(schedule.get\_classes()[j].get\_subject().get\_code()))  
 for j in range(0, len(schedule.get\_classes())):  
 if (schedule.get\_classes()[j].get\_subject().get\_numbHour() > schedule.get\_classes()[  
 j].get\_meetingTime().get\_MThour()):  
 if (schedule.get\_classes()[j].get\_section() == sections[i]):  
 instructorBlock.append(str(schedule.get\_classes()[j].get\_instructor().get\_number()))  
 timeBlock.append(str(schedule.get\_classes()[j].get\_meetingTime1().get\_id()))  
 roomBlock.append(str(schedule.get\_classes()[j].get\_room().get\_number()))  
 subjectBlock.append(str(schedule.get\_classes()[j].get\_subject().get\_code()))  
 for j in range(0, len(overtimeConflicts)):  
 if (overtimeConflicts[j].get\_section() == sections[i]):  
 xinstructorBlock.append(str(overtimeConflicts[j].get\_instructor().get\_number()))  
 xtimeBlock.append(str(overtimeConflicts[j].get\_meetingTime().get\_id()))  
 xroomBlock.append(str(overtimeConflicts[j].get\_room().get\_number()))  
 xsubjectBlock.append(str(overtimeConflicts[j].get\_subject().get\_code()))  
 for j in range(0, len(overtimeConflicts)):  
 if (overtimeConflicts[j].get\_subject().get\_numbHour() > overtimeConflicts[  
 j].get\_meetingTime().get\_MThour()):  
 if (overtimeConflicts[j].get\_section() == sections[i]):  
 xinstructorBlock.append(str(overtimeConflicts[j].get\_instructor().get\_number()))  
 xtimeBlock.append(str(overtimeConflicts[j].get\_meetingTime1().get\_id()))  
 xroomBlock.append(str(overtimeConflicts[j].get\_room().get\_number()))  
 xsubjectBlock.append(str(overtimeConflicts[j].get\_subject().get\_code()))  
 for j in range(0, len(overloadConflicts)):  
 if (overloadConflicts[j].get\_section() == sections[i]):  
 yinstructorBlock.append(str(overloadConflicts[j].get\_instructor().get\_number()))  
 ytimeBlock.append(str(overloadConflicts[j].get\_meetingTime().get\_id()))  
 yroomBlock.append(str(overloadConflicts[j].get\_room().get\_number()))  
 ysubjectBlock.append(str(overloadConflicts[j].get\_subject().get\_code()))  
 for j in range(0, len(overloadConflicts)):  
 if (overloadConflicts[j].get\_subject().get\_numbHour() > overloadConflicts[  
 j].get\_meetingTime().get\_MThour()):  
 if (overloadConflicts[j].get\_section() == sections[i]):  
 yinstructorBlock.append(str(overloadConflicts[j].get\_instructor().get\_number()))  
 ytimeBlock.append(str(overloadConflicts[j].get\_meetingTime1().get\_id()))  
 yroomBlock.append(str(overloadConflicts[j].get\_room().get\_number()))  
 ysubjectBlock.append(str(overloadConflicts[j].get\_subject().get\_code()))  
 section\_instructorBlocks.append(instructorBlock)  
 MT\_timeBlocks.append(timeBlock)  
 section\_rooms.append(roomBlock)  
 section\_subjects.append(subjectBlock)  
 xsection\_instructorBlocks.append(xinstructorBlock)  
 xMT\_timeBlocks.append(xtimeBlock)  
 xsection\_rooms.append(xroomBlock)  
 xsection\_subjects.append(xsubjectBlock)  
 ysection\_instructorBlocks.append(yinstructorBlock)  
 yMT\_timeBlocks.append(ytimeBlock)  
 ysection\_rooms.append(yroomBlock)  
 ysection\_subjects.append(ysubjectBlock)  
 timeCells=dbMgr.get\_meetingTimes()  
 meeetingTimes=list()  
 cells=list()  
 for i in range(0, len(timeCells)):  
 meeetingTimes.append(timeCells[i].get\_id())  
 cells.append(timeCells[i].get\_cell())  
  
 *# create file (workbook) and worksheet* outWorkbook=xlsxwriter.Workbook(**"xsection.xlsx"**)  
  
 def SECTIONBLOCKS():  
 *# declare data* columnData=[**"Time"**, **"Monday"**, **"Tuesday"**, **"Wednesday"**, **"Thursday"**, **"Friday"**, **"Saturday"**, **"Sunday"**]  
 timeBlock=[**"7:00 - 7:30"**, **"7:30 - 8:00"**, **"8:00 - 8:30"**, **"8:30 - 9:00"**, **"9:00 - 9:30"**,  
 **"9:30 - 10:00"**,  
 **"10:00 - 10:30"**,  
 **"10:30 - 11:00"**, **"11:00 - 11:30"**, **"11:30 - 12:00"**, **"12:00 - 12:30"**, **"12:30 - 1:00"**,  
 **"1:00 - 1:30"**,  
 **"1:30 -2:00"**, **"2:00 - 2:30"**, **"2:30 - 3:00"**, **"3:00 - 3:30"**, **"3:30 - 4:00"**,  
 **"4:00 - 4:30"**,  
 **"4:30 - 5:00"**,  
 **"5:00 - 5:30"**, **"5:30 - 6:00"**, **"6:00 - 6:30"**, **"6:30 - 7:00"**, **"7:00 - 7:30"**,  
 **"7:30 - 8:00"**,  
 **"8:00 - 8:30"**,  
 **"8:30 - 9:00"**]  
 load\_title=**'STUDENT SCHEDULE'** summaryColumn=[**"SUBJECT"**, **"TIME"**, **"DAY"**, **"ROOM"**, **"UNIT"**, **"# OF STUDENTS"**, **"FACULTY"**]  
 outputSheet=outWorkbook.add\_worksheet(name = sectionCodes[sheet])  
 *# Increase the cell size of the merged cells to highlight the formatting.* outputSheet.set\_column(**'A:H'**, 15)  
 outputSheet.set\_default\_row(20)  
 *# Create a format to use in the...  
 # ...centerBoldHead.* centerBoldHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...centerBoldHeadless.* centerBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...leftBoldHeadless.* leftBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...uleftHeadless.* uleftHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...rightBoldHeadless.* rightBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 0,  
 **'align'**: **'right'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...centerHead.* centerHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...ucenterHeadless.* ucenterHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...icenterHeadless.* icenterHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'italic'**: 1,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...smallcenterHead.* smallcenterHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 7.5,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...xsmallcenterHead.* xsmallcenterHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 7.5,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'fadfca'**})  
 *# ...ysmallcenterHead.* ysmallcenterHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 7.5,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'ffe338'**})  
 *# ...leftHeadless.* leftHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...details.* sheet\_details=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vbottom'**,  
 **'fg\_color'**: **'white'**})  
 *# ...header.* header=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 22,  
 **'font\_color'**: **'white'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'c31d3a'**}) *# Green, Accent 6, Darker 25%: 76933C  
 # ... columns.* column=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'white'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'c41e39'**}) *# Green, Accent 6: 9BBB59  
 # ...time.* time=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**}) *# Green, Accent 6, Lighter 40%: C4D798  
 # ...classSched.* classSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'ececec'**}) *# Emerald: 50C878  
 # ...xclassSched.* xclassSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'fadfca'**}) *# light 3 matte orange  
 # ...yclassSched.* yclassSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'ffe338'**}) *# Banana Yellow  
 # Write inside the Cell  
 # ...columnData* for item in range(len(columnData)):  
 outputSheet.write(11, item + 0, columnData[item], column)  
 *# ...timeBlock* for item in range(len(timeBlock)):  
 outputSheet.write(item + 12, 0, timeBlock[item], time)  
 *# ...Merge School* outputSheet.merge\_range(**'B1:F4'**, **''**, centerHead)  
 outputSheet.write\_rich\_string(**'B1'**, centerBoldHead, schooltext, centerHead, school\_infotext,  
 centerHead)  
 *# ...Load Title* outputSheet.merge\_range(**'B5:F6'**, load\_title, centerBoldHead)  
 *# ...VAA-OAP* outputSheet.merge\_range(**'A5:A6'**, vaa\_oap, centerHead)  
 *# ...Sheet Details* outputSheet.write\_string(**'G1'**, **'Index No.'**, sheet\_details)  
 outputSheet.write\_string(**'G2'**, **'Issue No.'**, sheet\_details)  
 outputSheet.write\_string(**'G3'**, **'Revision No.'**, sheet\_details)  
 outputSheet.write\_string(**'G4'**, **'Date'**, sheet\_details)  
 outputSheet.write\_string(**'G5'**, **'Page'**, sheet\_details)  
 outputSheet.write\_string(**'G6'**, **'QAC No.'**, sheet\_details)  
 outputSheet.write\_string(**'H1'**, str(indexno), sheet\_details)  
 outputSheet.write\_string(**'H2'**, str(issueno), sheet\_details)  
 outputSheet.write\_string(**'H3'**, str(revisionno), sheet\_details)  
 outputSheet.write\_string(**'H4'**, str(date), sheet\_details)  
 outputSheet.write\_string(**'H5'**, str(page), sheet\_details)  
 outputSheet.write\_string(**'H6'**, str(qacno), sheet\_details)  
 *# ...Logo* outputSheet.merge\_range(**'A1:A4'**, **''**, centerHead)  
 outputSheet.insert\_image(**'A1'**, **'tup-logo.png'**, {**'x\_scale'**: 1.1, **'y\_scale'**: 1.1})  
 *# ...College* outputSheet.merge\_range(**'A8:B8'**, **' COLLEGE:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C8:E8'**,  
 **'\_'** + college + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...Department* outputSheet.merge\_range(**'A9:B9'**, **' DEPARTMENT:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C9:E9'**,  
 **'\_'** + department + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...Name of Faculty* outputSheet.merge\_range(**'A10:B10'**, **' SECTION:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C10:E10'**,  
 **'\_'** + str(  
 sectionCodes[sheet]) + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...S.Y.* outputSheet.write\_string(**'G8'**, **'S.Y.:'**, rightBoldHeadless)  
 outputSheet.write\_string(**'H8'**, **'\_\_'** + schoolyear + **'\_\_'**, uleftHeadless)  
 *# ...Semester* outputSheet.write\_string(**'G9'**, **'Semester:'**, rightBoldHeadless)  
 outputSheet.write\_string(**'H9'**, **'\_'** + semester + **'\_'**, uleftHeadless)  
 *# ...bordercells* for row in range(12, 40):  
 for col in range(1, 8):  
 outputSheet.write(row, col, **''**, centerHead)  
 *# Summary Table  
 # ...summaryColumn* for item in range(len(summaryColumn)):  
 outputSheet.write(43, item, summaryColumn[item], column)  
 outputSheet.merge\_range(43, 6, 43, 7, summaryColumn[6], column)  
  
 for row in range(44, 56):  
 for col in range(0, 6):  
 outputSheet.write(row, col, **''**, centerHead)  
 outputSheet.merge\_range(row, 6, row, 7, **''**, centerHead)  
 classes=schedule.get\_classes()  
 class\_list=list()  
 overtimeConflicts\_list=list()  
 overloadConflicts\_list=list()  
 for i in range(0, len(classes)):  
 if (classes[i].get\_section().get\_code() == sectionCodes[sheet]):  
 class\_list.append(classes[i])  
 for i in range(0, len(overtimeConflicts)):  
 if (overtimeConflicts[i].get\_section().get\_code() == sectionCodes[sheet]):  
 overtimeConflicts\_list.append(overtimeConflicts[i])  
 for i in range(0, len(overloadConflicts)):  
 if (overloadConflicts[i].get\_section().get\_code() == sectionCodes[sheet]):  
 overloadConflicts\_list.append(overloadConflicts[i])  
 for i in range(0, len(class\_list)):  
 if class\_list[i].get\_section().get\_code() == sectionCodes[sheet]:  
 outputSheet.write(44 + i, 0, class\_list[i].get\_subject().get\_code(), smallcenterHead)  
 if class\_list[i].get\_meetingTime1().get\_id() != **'NULL'**:  
 outputSheet.write(44 + i, 1,  
 str(class\_list[i].get\_meetingTime().get\_time()) + **'**\n**'** + str(  
 class\_list[i].get\_meetingTime1().get\_time()), smallcenterHead)  
 outputSheet.write(44 + i, 2,  
 str(class\_list[i].get\_meetingTime().get\_day()[0:3]) + **'**\n**'** + str(  
 class\_list[i].get\_meetingTime1().get\_day()[0:3]), smallcenterHead)  
 else:  
 outputSheet.write(44 + i, 1, class\_list[i].get\_meetingTime().get\_time(),  
 smallcenterHead)  
 outputSheet.write(44 + i, 2, class\_list[i].get\_meetingTime().get\_day()[0:3],  
 smallcenterHead)  
 outputSheet.write(44 + i, 3, class\_list[i].get\_room().get\_number(), smallcenterHead)  
 outputSheet.write(44 + i, 4, class\_list[i].get\_subject().get\_units(), smallcenterHead)  
 outputSheet.write(44 + i, 5, class\_list[i].get\_subject().get\_maxNumbOfStudents(),  
 smallcenterHead)  
 outputSheet.write(44 + i, 6, class\_list[i].get\_instructor().get\_name(), smallcenterHead)  
 for i in range(0, len(overtimeConflicts\_list)):  
 if overtimeConflicts\_list[i].get\_section().get\_code() == sectionCodes[sheet]:  
 outputSheet.write(44 + i + len(class\_list), 0,  
 overtimeConflicts\_list[i].get\_subject().get\_code(), xsmallcenterHead)  
 if overtimeConflicts\_list[i].get\_meetingTime1().get\_id() != **'NULL'**:  
 outputSheet.write(44 + i + len(class\_list), 1, str(  
 overtimeConflicts\_list[i].get\_meetingTime().get\_time()) + **'**\n**'** + str(  
 overtimeConflicts\_list[i].get\_meetingTime1().get\_time()), xsmallcenterHead)  
 outputSheet.write(44 + i + len(class\_list), 2, str(  
 overtimeConflicts\_list[i].get\_meetingTime().get\_day()[0:3]) + **'**\n**'** + str(  
 overtimeConflicts\_list[i].get\_meetingTime1().get\_day()[0:3]), xsmallcenterHead)  
 else:  
 outputSheet.write(44 + i + len(class\_list), 1,  
 overtimeConflicts\_list[i].get\_meetingTime().get\_time(),  
 xsmallcenterHead)  
 outputSheet.write(44 + i + len(class\_list), 2,  
 overtimeConflicts\_list[i].get\_meetingTime().get\_day()[0:3],  
 xsmallcenterHead)  
 outputSheet.write(44 + i + len(class\_list), 3,  
 overtimeConflicts\_list[i].get\_room().get\_number(), xsmallcenterHead)  
 outputSheet.write(44 + i + len(class\_list), 4,  
 overtimeConflicts\_list[i].get\_subject().get\_units(), xsmallcenterHead)  
 outputSheet.write(44 + i + len(class\_list), 5,  
 overtimeConflicts\_list[i].get\_subject().get\_maxNumbOfStudents(),  
 xsmallcenterHead)  
 outputSheet.write(44 + i + len(class\_list), 6,  
 overtimeConflicts\_list[i].get\_instructor().get\_name(), xsmallcenterHead)  
 for i in range(0, len(overloadConflicts\_list)):  
 if overloadConflicts\_list[i].get\_section().get\_code() == sectionCodes[sheet]:  
 outputSheet.write(44 + i + len(class\_list) + len(overtimeConflicts\_list), 0,  
 overloadConflicts\_list[i].get\_subject().get\_code(), ysmallcenterHead)  
 if overloadConflicts\_list[i].get\_meetingTime1().get\_id() != **'NULL'**:  
 outputSheet.write(44 + i + len(class\_list) + len(overtimeConflicts\_list), 1,  
 str(overloadConflicts\_list[  
 i].get\_meetingTime().get\_time()) + **'**\n**'** + str(  
 overloadConflicts\_list[i].get\_meetingTime1().get\_time()),  
 ysmallcenterHead)  
 outputSheet.write(44 + i + len(class\_list) + len(overtimeConflicts\_list), 2, str(  
 overloadConflicts\_list[i].get\_meetingTime().get\_day()[0:3]) + **'**\n**'** + str(  
 overloadConflicts\_list[i].get\_meetingTime1().get\_day()[0:3]), ysmallcenterHead)  
 else:  
 outputSheet.write(44 + i + len(class\_list) + len(overtimeConflicts\_list), 1,  
 overloadConflicts\_list[i].get\_meetingTime().get\_time(),  
 ysmallcenterHead)  
 outputSheet.write(44 + i + len(class\_list) + len(overtimeConflicts\_list), 2,  
 overloadConflicts\_list[i].get\_meetingTime().get\_day()[0:3],  
 ysmallcenterHead)  
 outputSheet.write(44 + i + len(class\_list) + len(overtimeConflicts\_list), 3,  
 overloadConflicts\_list[i].get\_room().get\_number(), ysmallcenterHead)  
 outputSheet.write(44 + i + len(class\_list) + len(overtimeConflicts\_list), 4,  
 overloadConflicts\_list[i].get\_subject().get\_units(), ysmallcenterHead)  
 outputSheet.write(44 + i + len(class\_list) + len(overtimeConflicts\_list), 5,  
 overloadConflicts\_list[i].get\_subject().get\_maxNumbOfStudents(),  
 ysmallcenterHead)  
 outputSheet.write(44 + i + len(class\_list) + len(overtimeConflicts\_list), 6,  
 overloadConflicts\_list[i].get\_instructor().get\_name(), ysmallcenterHead)  
 *# ...Authority Sign* outputSheet.merge\_range(**'A57:C57'**, **' Prepared By:'**, leftHeadless)  
 outputSheet.merge\_range(**'A59:C59'**, **'\_\_\_\_\_'** + str(depthead) + **'\_\_\_\_\_'**, ucenterHeadless)  
 outputSheet.merge\_range(**'A60:C60'**, **'Department Head'**, icenterHeadless)  
 outputSheet.merge\_range(**'F57:H57'**, **' Recommending Approval:'**, leftHeadless)  
 outputSheet.merge\_range(**'F59:H59'**, **'\_\_\_\_\_'** + str(dean) + **'\_\_\_\_\_'**, ucenterHeadless)  
 outputSheet.merge\_range(**'F60:H60'**, **'College Dean'**, icenterHeadless)  
 *# ...footer* outputSheet.write\_string(**'B62'**, **'Transaction ID'**, sheet\_details)  
 outputSheet.merge\_range(**'C62:G62'**, **''**, sheet\_details)  
 outputSheet.write\_string(**'B63'**, **'Signature'**, sheet\_details)  
 outputSheet.merge\_range(**'C63:G63'**, **''**, sheet\_details)  
  
 for MT in MT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in MT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 section\_instructorBlocks[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 section\_rooms[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 section\_subjects[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 MT\_timeBlocks[sheet][MT\_timeBlocks[sheet].index(MT)]), classSched)  
 for MT in xMT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in xMT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 xsection\_instructorBlocks[sheet][xMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 xsection\_rooms[sheet][xMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 xsection\_subjects[sheet][xMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 xMT\_timeBlocks[sheet][xMT\_timeBlocks[sheet].index(MT)]), xclassSched)  
 for MT in yMT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in yMT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 ysection\_instructorBlocks[sheet][yMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 ysection\_rooms[sheet][yMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 ysection\_subjects[sheet][yMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 yMT\_timeBlocks[sheet][yMT\_timeBlocks[sheet].index(MT)]), yclassSched)  
  
 for numb\_of\_sections in range(0, len(sections)):  
 sheet=numb\_of\_sections  
 SECTIONBLOCKS()  
 outWorkbook.close()  
  
 @staticmethod  
 def XROOMXLSX(schedule):  
 rooms=dbMgr.get\_rooms()  
 roomNumbers=list()  
 room\_sectionBlocks=list()  
 MT\_timeBlocks=list()  
 room\_instructors=list()  
 room\_subjects=list()  
 xroom\_sectionBlocks=list()  
 xMT\_timeBlocks=list()  
 xroom\_instructors=list()  
 xroom\_subjects=list()  
 yroom\_sectionBlocks=list()  
 yMT\_timeBlocks=list()  
 yroom\_instructors=list()  
 yroom\_subjects=list()  
 for i in range(0, len(rooms)):  
 roomNumbers.append(rooms[i].get\_number())  
 sectionBlock=list()  
 instructorBlock=list()  
 subjectBlock=list()  
 timeBlock=list()  
 xsectionBlock=list()  
 xinstructorBlock=list()  
 xsubjectBlock=list()  
 xtimeBlock=list()  
 ysectionBlock=list()  
 yinstructorBlock=list()  
 ysubjectBlock=list()  
 ytimeBlock=list()  
 for j in range(0, len(schedule.get\_classes())):  
 if (schedule.get\_classes()[j].get\_room() == rooms[i]):  
 sectionBlock.append(str(schedule.get\_classes()[j].get\_section().get\_code()))  
 instructorBlock.append(str(schedule.get\_classes()[j].get\_instructor().get\_number()))  
 subjectBlock.append(str(schedule.get\_classes()[j].get\_subject().get\_code()))  
 timeBlock.append(str(schedule.get\_classes()[j].get\_meetingTime().get\_id()))  
 for j in range(0, len(schedule.get\_classes())):  
 if (schedule.get\_classes()[j].get\_subject().get\_numbHour() > schedule.get\_classes()[  
 j].get\_meetingTime().get\_MThour()):  
 if (schedule.get\_classes()[j].get\_room() == rooms[i]):  
 sectionBlock.append(str(schedule.get\_classes()[j].get\_section().get\_code()))  
 instructorBlock.append(str(schedule.get\_classes()[j].get\_instructor().get\_number()))  
 subjectBlock.append(str(schedule.get\_classes()[j].get\_subject().get\_code()))  
 timeBlock.append(str(schedule.get\_classes()[j].get\_meetingTime1().get\_id()))  
 for j in range(0, len(overtimeConflicts)):  
 if (overtimeConflicts[j].get\_room() == rooms[i]):  
 xsectionBlock.append(overtimeConflicts[j].get\_section().get\_code())  
 xinstructorBlock.append(overtimeConflicts[j].get\_instructor().get\_number())  
 xsubjectBlock.append(overtimeConflicts[j].get\_subject().get\_code())  
 xtimeBlock.append(str(overtimeConflicts[j].get\_meetingTime().get\_id()))  
 for j in range(0, len(overtimeConflicts)):  
 if (overtimeConflicts[j].get\_subject().get\_numbHour() > overtimeConflicts[  
 j].get\_meetingTime().get\_MThour()):  
 if (overtimeConflicts[j].get\_room() == rooms[i]):  
 xsectionBlock.append(str(overtimeConflicts[j].get\_section().get\_code()))  
 xinstructorBlock.append(str(overtimeConflicts[j].get\_instructor().get\_number()))  
 xsubjectBlock.append(str(overtimeConflicts[j].get\_subject().get\_code()))  
 xtimeBlock.append(str(overtimeConflicts[j].get\_meetingTime1().get\_id()))  
 for j in range(0, len(overloadConflicts)):  
 if (overloadConflicts[j].get\_room() == rooms[i]):  
 ysectionBlock.append(overloadConflicts[j].get\_section().get\_code())  
 yinstructorBlock.append(overloadConflicts[j].get\_instructor().get\_number())  
 ysubjectBlock.append(overloadConflicts[j].get\_subject().get\_code())  
 ytimeBlock.append(str(overloadConflicts[j].get\_meetingTime().get\_id()))  
 for j in range(0, len(overloadConflicts)):  
 if (overloadConflicts[j].get\_subject().get\_numbHour() > overloadConflicts[  
 j].get\_meetingTime().get\_MThour()):  
 if (overloadConflicts[j].get\_room() == rooms[i]):  
 ysectionBlock.append(str(overloadConflicts[j].get\_section().get\_code()))  
 yinstructorBlock.append(str(overloadConflicts[j].get\_instructor().get\_number()))  
 ysubjectBlock.append(str(overloadConflicts[j].get\_subject().get\_code()))  
 ytimeBlock.append(str(overloadConflicts[j].get\_meetingTime1().get\_id()))  
 room\_sectionBlocks.append(sectionBlock)  
 room\_instructors.append(instructorBlock)  
 room\_subjects.append(subjectBlock)  
 MT\_timeBlocks.append(timeBlock)  
 xroom\_sectionBlocks.append(xsectionBlock)  
 xroom\_instructors.append(xinstructorBlock)  
 xroom\_subjects.append(xsubjectBlock)  
 xMT\_timeBlocks.append(xtimeBlock)  
 yroom\_sectionBlocks.append(ysectionBlock)  
 yroom\_instructors.append(yinstructorBlock)  
 yroom\_subjects.append(ysubjectBlock)  
 yMT\_timeBlocks.append(ytimeBlock)  
 timeCells=dbMgr.get\_meetingTimes()  
 meeetingTimes=list()  
 cells=list()  
 for i in range(0, len(timeCells)):  
 meeetingTimes.append(timeCells[i].get\_id())  
 cells.append(timeCells[i].get\_cell())  
  
 *# create file (workbook) and worksheet* outWorkbook=xlsxwriter.Workbook(**"xroom.xlsx"**)  
  
 def ROOMBLOCKS():  
 *# declare data* columnData=[**"Time"**, **"Monday"**, **"Tuesday"**, **"Wednesday"**, **"Thursday"**, **"Friday"**, **"Saturday"**, **"Sunday"**]  
 timeBlock=[**"7:00 - 7:30"**, **"7:30 - 8:00"**, **"8:00 - 8:30"**, **"8:30 - 9:00"**, **"9:00 - 9:30"**,  
 **"9:30 - 10:00"**,  
 **"10:00 - 10:30"**,  
 **"10:30 - 11:00"**, **"11:00 - 11:30"**, **"11:30 - 12:00"**, **"12:00 - 12:30"**, **"12:30 - 1:00"**,  
 **"1:00 - 1:30"**,  
 **"1:30 -2:00"**, **"2:00 - 2:30"**, **"2:30 - 3:00"**, **"3:00 - 3:30"**, **"3:30 - 4:00"**,  
 **"4:00 - 4:30"**,  
 **"4:30 - 5:00"**,  
 **"5:00 - 5:30"**, **"5:30 - 6:00"**, **"6:00 - 6:30"**, **"6:30 - 7:00"**, **"7:00 - 7:30"**,  
 **"7:30 - 8:00"**,  
 **"8:00 - 8:30"**,  
 **"8:30 - 9:00"**]  
 load\_title=**'ROOM SCHEDULE'** outputSheet=outWorkbook.add\_worksheet(name = roomNumbers[sheet])  
 *# Increase the cell size of the merged cells to highlight the formatting.* outputSheet.set\_column(**'A:H'**, 15)  
 outputSheet.set\_default\_row(20)  
 *# Create a format to use in the...  
 # ...centerBoldHead.* centerBoldHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...centerBoldHeadless.* centerBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...leftBoldHeadless.* leftBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...uleftHeadless.* uleftHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...rightBoldHeadless.* rightBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 0,  
 **'align'**: **'right'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...centerHead.* centerHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...ucenterHeadless.* ucenterHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...icenterHeadless.* icenterHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'italic'**: 1,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...smallcenterHead.* smallcenterHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 7.5,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...leftHeadless.* leftHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...details.* sheet\_details=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vbottom'**,  
 **'fg\_color'**: **'white'**})  
 *# ...header.* header=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 22,  
 **'font\_color'**: **'white'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'c31d3a'**}) *# Green, Accent 6, Darker 25%: 76933C  
 # ... columns.* column=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'white'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'c41e39'**}) *# Green, Accent 6: 9BBB59  
 # ...time.* time=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**}) *# Green, Accent 6, Lighter 40%: C4D798  
 # ...classSched.* classSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'ececec'**}) *# Emerald: 50C878  
 # ...xclassSched.* xclassSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'fadfca'**}) *# light 3 matte orange  
 # ...yclassSched.* yclassSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'ffe338'**}) *# Banana Yellow  
 # Write inside the Cell  
 # ...columnData* for item in range(len(columnData)):  
 outputSheet.write(11, item + 0, columnData[item], column)  
 *# ...timeBlock* for item in range(len(timeBlock)):  
 outputSheet.write(item + 12, 0, timeBlock[item], time)  
 *# ...Merge School* outputSheet.merge\_range(**'B1:F4'**, **''**, centerHead)  
 outputSheet.write\_rich\_string(**'B1'**, centerBoldHead, schooltext, centerHead, school\_infotext,  
 centerHead)  
 *# ...Load Title* outputSheet.merge\_range(**'B5:F6'**, load\_title, centerBoldHead)  
 *# ...VAA-OAP* outputSheet.merge\_range(**'A5:A6'**, vaa\_oap, centerHead)  
 *# ...Sheet Details* outputSheet.write\_string(**'G1'**, **'Index No.'**, sheet\_details)  
 outputSheet.write\_string(**'G2'**, **'Issue No.'**, sheet\_details)  
 outputSheet.write\_string(**'G3'**, **'Revision No.'**, sheet\_details)  
 outputSheet.write\_string(**'G4'**, **'Date'**, sheet\_details)  
 outputSheet.write\_string(**'G5'**, **'Page'**, sheet\_details)  
 outputSheet.write\_string(**'G6'**, **'QAC No.'**, sheet\_details)  
 outputSheet.write\_string(**'H1'**, str(indexno), sheet\_details)  
 outputSheet.write\_string(**'H2'**, str(issueno), sheet\_details)  
 outputSheet.write\_string(**'H3'**, str(revisionno), sheet\_details)  
 outputSheet.write\_string(**'H4'**, str(date), sheet\_details)  
 outputSheet.write\_string(**'H5'**, str(page), sheet\_details)  
 outputSheet.write\_string(**'H6'**, str(qacno), sheet\_details)  
 *# ...Logo* outputSheet.merge\_range(**'A1:A4'**, **''**, centerHead)  
 outputSheet.insert\_image(**'A1'**, **'tup-logo.png'**, {**'x\_scale'**: 1.1, **'y\_scale'**: 1.1})  
 *# ...College* outputSheet.merge\_range(**'A8:B8'**, **' COLLEGE:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C8:E8'**,  
 **'\_'** + college + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...Department* outputSheet.merge\_range(**'A9:B9'**, **' DEPARTMENT:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C9:E9'**,  
 **'\_'** + department + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...Name of Faculty* outputSheet.merge\_range(**'A10:B10'**, **' ROOM:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C10:E10'**,  
 **'\_'** + str(  
 roomNumbers[sheet]) + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...S.Y.* outputSheet.write\_string(**'G8'**, **'S.Y.:'**, rightBoldHeadless)  
 outputSheet.write\_string(**'H8'**, **'\_\_'** + schoolyear + **'\_\_'**, uleftHeadless)  
 *# ...Semester* outputSheet.write\_string(**'G9'**, **'Semester:'**, rightBoldHeadless)  
 outputSheet.write\_string(**'H9'**, **'\_'** + semester + **'\_'**, uleftHeadless)  
 *# ...bordercells* for row in range(12, 40):  
 for col in range(1, 8):  
 outputSheet.write(row, col, **''**, centerHead)  
 *# ...Authority Sign* outputSheet.merge\_range(**'A44:C44'**, **' Prepared By:'**, leftHeadless)  
 outputSheet.merge\_range(**'A46:C46'**, **'\_\_\_\_\_'** + str(depthead) + **'\_\_\_\_\_'**, ucenterHeadless)  
 outputSheet.merge\_range(**'A47:C47'**, **'Department Head'**, icenterHeadless)  
 outputSheet.merge\_range(**'F44:H44'**, **' Recommending Approval:'**, leftHeadless)  
 outputSheet.merge\_range(**'F46:H46'**, **'\_\_\_\_\_'** + str(dean) + **'\_\_\_\_\_'**, ucenterHeadless)  
 outputSheet.merge\_range(**'F47:H47'**, **'College Dean'**, icenterHeadless)  
 *# ...footer* outputSheet.write\_string(**'B50'**, **'Transaction ID'**, sheet\_details)  
 outputSheet.merge\_range(**'C50:G50'**, **''**, sheet\_details)  
 outputSheet.write\_string(**'B51'**, **'Signature'**, sheet\_details)  
 outputSheet.merge\_range(**'C51:G51'**, **''**, sheet\_details)  
  
 for MT in MT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in MT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 room\_instructors[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 room\_sectionBlocks[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 room\_subjects[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 MT\_timeBlocks[sheet][MT\_timeBlocks[sheet].index(MT)]), classSched)  
 for MT in xMT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in xMT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 xroom\_instructors[sheet][xMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 xroom\_sectionBlocks[sheet][xMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 xroom\_subjects[sheet][xMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 xMT\_timeBlocks[sheet][xMT\_timeBlocks[sheet].index(MT)]), xclassSched)  
 for MT in yMT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in yMT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 yroom\_instructors[sheet][yMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 yroom\_sectionBlocks[sheet][yMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 yroom\_subjects[sheet][yMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 yMT\_timeBlocks[sheet][yMT\_timeBlocks[sheet].index(MT)]), yclassSched)  
  
 for numb\_of\_rooms in range(0, len(rooms)):  
 sheet=numb\_of\_rooms  
 ROOMBLOCKS()  
 outWorkbook.close()  
  
 @staticmethod  
 def XINSTRUCTORXLSX(schedule):  
 instructors=dbMgr.get\_instructors()  
 instructorIDs=list()  
 instructor\_sectionBlocks=list()  
 MT\_timeBlocks=list()  
 instructor\_rooms=list()  
 instructor\_subjects=list()  
 xinstructor\_sectionBlocks=list()  
 xMT\_timeBlocks=list()  
 xinstructor\_rooms=list()  
 xinstructor\_subjects=list()  
 yinstructor\_sectionBlocks=list()  
 yMT\_timeBlocks=list()  
 yinstructor\_rooms=list()  
 yinstructor\_subjects=list()  
 for i in range(0, len(instructors)):  
 instructorIDs.append(instructors[i].get\_name())  
 sectionBlock=list()  
 timeBlock=list()  
 roomBlock=list()  
 subjectBlock=list()  
 xsectionBlock=list()  
 xtimeBlock=list()  
 xroomBlock=list()  
 xsubjectBlock=list()  
 ysectionBlock=list()  
 ytimeBlock=list()  
 yroomBlock=list()  
 ysubjectBlock=list()  
 for j in range(0, len(schedule.get\_classes())):  
 if (schedule.get\_classes()[j].get\_instructor() == instructors[i]):  
 sectionBlock.append(str(schedule.get\_classes()[j].get\_section().get\_code()))  
 timeBlock.append(str(schedule.get\_classes()[j].get\_meetingTime().get\_id()))  
 roomBlock.append(str(schedule.get\_classes()[j].get\_room().get\_number()))  
 subjectBlock.append(str(schedule.get\_classes()[j].get\_subject().get\_code()))  
 for j in range(0, len(schedule.get\_classes())):  
 if (schedule.get\_classes()[j].get\_subject().get\_numbHour() > schedule.get\_classes()[  
 j].get\_meetingTime().get\_MThour()):  
 if (schedule.get\_classes()[j].get\_instructor() == instructors[i]):  
 sectionBlock.append(str(schedule.get\_classes()[j].get\_section().get\_code()))  
 timeBlock.append(str(schedule.get\_classes()[j].get\_meetingTime1().get\_id()))  
 roomBlock.append(str(schedule.get\_classes()[j].get\_room().get\_number()))  
 subjectBlock.append(str(schedule.get\_classes()[j].get\_subject().get\_code()))  
 for j in range(0, len(overtimeConflicts)):  
 if (overtimeConflicts[j].get\_instructor() == instructors[i]):  
 xsectionBlock.append(str(overtimeConflicts[j].get\_section().get\_code()))  
 xtimeBlock.append(str(overtimeConflicts[j].get\_meetingTime().get\_id()))  
 xroomBlock.append(str(overtimeConflicts[j].get\_room().get\_number()))  
 xsubjectBlock.append(str(overtimeConflicts[j].get\_subject().get\_code()))  
 for j in range(0, len(overtimeConflicts)):  
 if (overtimeConflicts[j].get\_subject().get\_numbHour() > overtimeConflicts[  
 j].get\_meetingTime().get\_MThour()):  
 if (overtimeConflicts[j].get\_instructor() == instructors[i]):  
 xsectionBlock.append(overtimeConflicts[j].get\_section().get\_code())  
 xtimeBlock.append(str(overtimeConflicts[j].get\_meetingTime1().get\_id()))  
 xroomBlock.append(str(overtimeConflicts[j].get\_room().get\_number()))  
 xsubjectBlock.append(str(overtimeConflicts[j].get\_subject().get\_code()))  
 for j in range(0, len(overloadConflicts)):  
 if (overloadConflicts[j].get\_instructor() == instructors[i]):  
 ysectionBlock.append(str(overloadConflicts[j].get\_section().get\_code()))  
 ytimeBlock.append(str(overloadConflicts[j].get\_meetingTime().get\_id()))  
 yroomBlock.append(str(overloadConflicts[j].get\_room().get\_number()))  
 ysubjectBlock.append(str(overloadConflicts[j].get\_subject().get\_code()))  
 for j in range(0, len(overloadConflicts)):  
 if (overloadConflicts[j].get\_subject().get\_numbHour() > overloadConflicts[  
 j].get\_meetingTime().get\_MThour()):  
 if (overloadConflicts[j].get\_instructor() == instructors[i]):  
 ysectionBlock.append(overloadConflicts[j].get\_section().get\_code())  
 ytimeBlock.append(str(overloadConflicts[j].get\_meetingTime1().get\_id()))  
 yroomBlock.append(str(overloadConflicts[j].get\_room().get\_number()))  
 ysubjectBlock.append(str(overloadConflicts[j].get\_subject().get\_code()))  
 instructor\_sectionBlocks.append(sectionBlock)  
 MT\_timeBlocks.append(timeBlock)  
 instructor\_rooms.append(roomBlock)  
 instructor\_subjects.append(subjectBlock)  
 xinstructor\_sectionBlocks.append(xsectionBlock)  
 xMT\_timeBlocks.append(xtimeBlock)  
 xinstructor\_rooms.append(xroomBlock)  
 xinstructor\_subjects.append(xsubjectBlock)  
 yinstructor\_sectionBlocks.append(ysectionBlock)  
 yMT\_timeBlocks.append(ytimeBlock)  
 yinstructor\_rooms.append(yroomBlock)  
 yinstructor\_subjects.append(ysubjectBlock)  
 timeCells=dbMgr.get\_meetingTimes()  
 meeetingTimes=list()  
 cells=list()  
 for i in range(0, len(timeCells)):  
 meeetingTimes.append(timeCells[i].get\_id())  
 cells.append(timeCells[i].get\_cell())  
  
 *# create file (workbook) and worksheet* outWorkbook=xlsxwriter.Workbook(**"xinstructor.xlsx"**)  
  
 def INSTRUCTORBLOCKS():  
 *# declare data* columnData=[**"Time"**, **"Monday"**, **"Tuesday"**, **"Wednesday"**, **"Thursday"**, **"Friday"**, **"Saturday"**, **"Sunday"**]  
 timeBlock=[**"7:00 - 7:30"**, **"7:30 - 8:00"**, **"8:00 - 8:30"**, **"8:30 - 9:00"**, **"9:00 - 9:30"**,  
 **"9:30 - 10:00"**,  
 **"10:00 - 10:30"**,  
 **"10:30 - 11:00"**, **"11:00 - 11:30"**, **"11:30 - 12:00"**, **"12:00 - 12:30"**, **"12:30 - 1:00"**,  
 **"1:00 - 1:30"**,  
 **"1:30 -2:00"**, **"2:00 - 2:30"**, **"2:30 - 3:00"**, **"3:00 - 3:30"**, **"3:30 - 4:00"**,  
 **"4:00 - 4:30"**,  
 **"4:30 - 5:00"**,  
 **"5:00 - 5:30"**, **"5:30 - 6:00"**, **"6:00 - 6:30"**, **"6:30 - 7:00"**, **"7:00 - 7:30"**,  
 **"7:30 - 8:00"**,  
 **"8:00 - 8:30"**,  
 **"8:30 - 9:00"**]  
 load\_title=**'INDIVIDUAL FACULTY LOADING'** outputSheet=outWorkbook.add\_worksheet(name = instructorIDs[sheet])  
 *# Increase the cell size of the merged cells to highlight the formatting.* outputSheet.set\_column(**'A:H'**, 15)  
 outputSheet.set\_default\_row(20)  
 *# Create a format to use in the...  
 # ...centerBoldHead.* centerBoldHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...centerBoldHeadless.* centerBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...leftBoldHeadless.* leftBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...uleftHeadless.* uleftHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...rightBoldHeadless.* rightBoldHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 11,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 1,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 0,  
 **'align'**: **'right'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...centerHead.* centerHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...ucenterHeadless.* ucenterHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 0,  
 **'underline'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...icenterHeadless.* icenterHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'italic'**: 1,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...smallcenterHead.* smallcenterHead=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 7.5,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'text\_wrap'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...leftHeadless.* leftHeadless=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 0,  
 **'text\_wrap'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**})  
 *# ...details.* sheet\_details=outWorkbook.add\_format({  
 **'font'**: **'Arial'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'align'**: **'left'**,  
 **'valign'**: **'vbottom'**,  
 **'fg\_color'**: **'white'**})  
 *# ...header.* header=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 22,  
 **'font\_color'**: **'white'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'c31d3a'**}) *# Green, Accent 6, Darker 25%: 76933C  
 # ... columns.* column=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'white'**,  
 **'bold'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'c41e39'**}) *# Green, Accent 6: 9BBB59  
 # ...time.* time=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 12,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'white'**}) *# Green, Accent 6, Lighter 40%: C4D798  
 # ...classSched.* classSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'ececec'**}) *# Emerald: 50C878  
 # ...xclassSched.* xclassSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'fadfca'**}) *# light 3 matte orange  
 # ...yclassSched.* yclassSched=outWorkbook.add\_format({  
 **'font'**: **'Calibri'**,  
 **'font\_size'**: 10,  
 **'font\_color'**: **'black'**,  
 **'bold'**: 0,  
 **'text\_wrap'**: 1,  
 **'border'**: 1,  
 **'align'**: **'center'**,  
 **'valign'**: **'vcenter'**,  
 **'fg\_color'**: **'ffe338'**}) *# Banana Yellow  
 # Write inside the Cell  
 # ...columnData* for item in range(len(columnData)):  
 outputSheet.write(11, item + 0, columnData[item], column)  
 *# ...timeBlock* for item in range(len(timeBlock)):  
 outputSheet.write(item + 12, 0, timeBlock[item], time)  
 *# ...Merge School* outputSheet.merge\_range(**'B1:F4'**, **''**, centerHead)  
 outputSheet.write\_rich\_string(**'B1'**, centerBoldHead, schooltext, centerHead, school\_infotext,  
 centerHead)  
 *# ...Load Title* outputSheet.merge\_range(**'B5:F6'**, load\_title, centerBoldHead)  
 *# ...VAA-OAP* outputSheet.merge\_range(**'A5:A6'**, vaa\_oap, centerHead)  
 *# ...Sheet Details* outputSheet.write\_string(**'G1'**, **'Index No.'**, sheet\_details)  
 outputSheet.write\_string(**'G2'**, **'Issue No.'**, sheet\_details)  
 outputSheet.write\_string(**'G3'**, **'Revision No.'**, sheet\_details)  
 outputSheet.write\_string(**'G4'**, **'Date'**, sheet\_details)  
 outputSheet.write\_string(**'G5'**, **'Page'**, sheet\_details)  
 outputSheet.write\_string(**'G6'**, **'QAC No.'**, sheet\_details)  
 outputSheet.write\_string(**'H1'**, str(indexno), sheet\_details)  
 outputSheet.write\_string(**'H2'**, str(issueno), sheet\_details)  
 outputSheet.write\_string(**'H3'**, str(revisionno), sheet\_details)  
 outputSheet.write\_string(**'H4'**, str(date), sheet\_details)  
 outputSheet.write\_string(**'H5'**, str(page), sheet\_details)  
 outputSheet.write\_string(**'H6'**, str(qacno), sheet\_details)  
 *# ...Logo* outputSheet.merge\_range(**'A1:A4'**, **''**, centerHead)  
 outputSheet.insert\_image(**'A1'**, **'tup-logo.png'**, {**'x\_scale'**: 1.1, **'y\_scale'**: 1.1})  
 *# ...College* outputSheet.merge\_range(**'A8:B8'**, **' COLLEGE:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C8:E8'**,  
 **'\_'** + college + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...Department* outputSheet.merge\_range(**'A9:B9'**, **' DEPARTMENT:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C9:E9'**,  
 **'\_'** + department + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...Name of Faculty* outputSheet.merge\_range(**'A10:B10'**, **' NAME OF FACULTY:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'C10:E10'**,  
 **'\_'** + str(  
 instructorIDs[sheet]) + **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**,  
 uleftHeadless)  
 *# ...S.Y.* outputSheet.write\_string(**'G8'**, **'S.Y.:'**, rightBoldHeadless)  
 outputSheet.write\_string(**'H8'**, **'\_\_'** + schoolyear + **'\_\_'**, uleftHeadless)  
 *# ...Semester* outputSheet.write\_string(**'G9'**, **'Semester:'**, rightBoldHeadless)  
 outputSheet.write\_string(**'H9'**, **'\_'** + semester + **'\_'**, uleftHeadless)  
 *# ...bordercells* for row in range(12, 40):  
 for col in range(1, 8):  
 outputSheet.write(row, col, **''**, centerHead)  
 *# ...below table* outputSheet.merge\_range(**'A44:B44'**, **' OFFICIAL TIME:'**, leftBoldHeadless)  
 outputSheet.merge\_range(**'A47:B47'**, **' OVERLOAD:'**, leftBoldHeadless)  
 outputSheet.write\_string(**'C43'**, **'DAY'**, centerBoldHeadless)  
 outputSheet.write\_string(**'C44'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.write\_string(**'C45'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.write\_string(**'C46'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.write\_string(**'C47'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.write\_string(**'C48'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.write\_string(**'C49'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.merge\_range(**'E43:F43'**, **'TIME'**, centerBoldHeadless)  
 outputSheet.merge\_range(**'E44:F44'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.merge\_range(**'E45:F45'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.merge\_range(**'E46:F46'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.merge\_range(**'E47:F47'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.merge\_range(**'E48:F48'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 outputSheet.merge\_range(**'E49:F49'**, **'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**, uleftHeadless)  
 *# ...footer* outputSheet.write\_string(**'B50'**, **'Transaction ID'**, sheet\_details)  
 outputSheet.merge\_range(**'C50:G50'**, **''**, sheet\_details)  
 outputSheet.write\_string(**'B51'**, **'Signature'**, sheet\_details)  
 outputSheet.merge\_range(**'C51:G51'**, **''**, sheet\_details)  
  
 for MT in MT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in MT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 instructor\_sectionBlocks[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 instructor\_subjects[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 instructor\_rooms[sheet][MT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 MT\_timeBlocks[sheet][MT\_timeBlocks[sheet].index(MT)]), classSched)  
 for MT in xMT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in xMT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 xinstructor\_sectionBlocks[sheet][xMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 xinstructor\_subjects[sheet][xMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 xinstructor\_rooms[sheet][xMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 xMT\_timeBlocks[sheet][xMT\_timeBlocks[sheet].index(MT)]), xclassSched)  
 for MT in yMT\_timeBlocks[sheet]:  
 for i in range(0, len(meeetingTimes)):  
 if (MT == meeetingTimes[i]):  
 MTBlock=str(cells[i])  
 if (MT in yMT\_timeBlocks[sheet]):  
 outputSheet.merge\_range(MTBlock, (  
 yinstructor\_sectionBlocks[sheet][yMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 yinstructor\_subjects[sheet][yMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 yinstructor\_rooms[sheet][yMT\_timeBlocks[sheet].index(MT)] + **'**\n**'** +  
 yMT\_timeBlocks[sheet][yMT\_timeBlocks[sheet].index(MT)]), yclassSched)  
  
 for numb\_of\_instructors in range(0, len(instructors)):  
 sheet=numb\_of\_instructors  
 INSTRUCTORBLOCKS()  
 outWorkbook.close()  
  
 @staticmethod  
 def display\_schedule\_meetingTimes(schedule):  
 print(**"> from 'meeting time' perspective"**)  
 meetingTimesTable=prettytable.PrettyTable(  
 [**'id'**, **'meeting time'**, **'classes [section,class,room,instructor,meeting-time] '**])  
 meetingTimes=dbMgr.get\_meetingTimes()  
 for i in range(0, len(meetingTimes)):  
 classes=list()  
 for j in range(0, len(schedule.get\_classes())):  
 if schedule.get\_classes()[j].get\_meetingTime() == meetingTimes[i]:  
 classes.append(str(schedule.get\_classes()[j]))  
 meetingTimesTable.add\_row([meetingTimes[i].get\_id(), meetingTimes[i].get\_time(), str(classes)])  
 print(meetingTimesTable)  
  
 @staticmethod  
 def display\_schedule\_rooms(schedule):  
 print(**"> from 'room' perspective"**)  
 scheduleRoomsTable=prettytable.PrettyTable(  
 [**'room'**, **'classes [section,class,room,instructor,meeting-time] '**])  
 rooms=dbMgr.get\_rooms()  
 for i in range(0, len(rooms)):  
 roomSchedule=list()  
 for j in range(0, len(schedule.get\_classes())):  
 if schedule.get\_classes()[j].get\_room() == rooms[i]:  
 roomSchedule.append(str(schedule.get\_classes()[j]))  
 scheduleRoomsTable.add\_row([str(rooms[i].get\_number()), str(roomSchedule)])  
 print(scheduleRoomsTable)  
  
 @staticmethod  
 def display\_schedule\_instructors(schedule):  
 print(**"> from 'instructor' perspective"**)  
 instructorsTable=prettytable.PrettyTable(  
 [**'ID'**, **'Instructor'**, **"Classes [section,class,room,instructor,meeting-time]"**,  
 **'Remaining Availability'**])  
 instructors=dbMgr.get\_instructors()  
 for i in range(0, len(instructors)):  
 availability=[]  
 for j in range(0, len(instructors[i].get\_availability())):  
 availability.append(instructors[i].get\_availability()[j])  
 classSchedule=list()  
 for j in range(0, len(schedule.get\_classes())):  
 if schedule.get\_classes()[j].get\_instructor() == instructors[i]:  
 classSchedule.append(str(schedule.get\_classes()[j]))  
 instructorsTable.add\_row(  
 [instructors[i].get\_number(), instructors[i].get\_name(), str(classSchedule), availability])  
 print(instructorsTable)  
  
 @staticmethod  
 def display\_schedule\_conflicts(schedule):  
 conflictsTable=prettytable.PrettyTable([**'conflict type'**, **'conflict between classes'**])  
 conflicts=schedule.get\_conflicts()  
 for i in range(0, len(conflicts)):  
 conflictsTable.add\_row([str(conflicts[i].get\_conflictType()),  
 str(**" and "**.join(map(str, conflicts[i].get\_conflictBetweenClasses())))])  
 if (len(conflicts) > 0): print(conflictsTable)  
  
 dbMgr=DBMgr()  
  
 sys.stdout=StdoutRedirector(textout)  
 ClearSTD()  
 DisplayMgr.display\_input\_data()  
 print(**'**\n**'**)  
 txt\_result.config(text = **"Successfully viewed Initial Data"**, fg = **"Black"**)  
  
  
 def ARIMA():  
 btn\_predict\_population[**'state'**] = DISABLED  
  
 txt\_result.config(text = **"Predict Population Button Clicked!"**, fg = **"#EE7600"**)  
  
 class DBMgr:  
 def \_\_init\_\_(self):  
 self.\_conn=sqlite.connect(**'ARIMADB.db'**) *# sql connector* self.\_cursor=self.\_conn.cursor() *# sql cursor* self.\_populations=self.\_select\_populations() *# select population sql cursor  
  
 # select population sql command* def \_select\_populations(self):  
 self.\_cursor.execute(**"SELECT \* FROM population"**)  
 populations=self.\_cursor.fetchall()  
 returnPopulations=[]  
 for i in range(0, len(populations)):  
 returnPopulations.append(Population(populations[i][0], populations[i][1]))  
 return returnPopulations  
  
 *# GETTING VALUE* def get\_populations(self):  
 return self.\_populations *# get item to population table* class Population:  
 def \_\_init\_\_(self, semester, number):  
 self.\_semester=semester  
 self.\_number=number  
  
 def get\_semester(self): return self.\_semester  
  
 def get\_number(self): return self.\_number  
  
 dbMgr=DBMgr()  
 sys.stdout=StdoutRedirector(textout)  
 ClearSTD()  
 populations=dbMgr.get\_populations()  
 semester=[]  
 number=[]  
 for i in range(0, len(populations)):  
 semester.append(populations[i].get\_semester())  
 number.append(populations[i].get\_number())  
 try:  
 os.remove(**'Students Pop.csv'**)  
 except FileNotFoundError:  
 pass  
 with open(**'Students Pop.csv'**, **'w'**, newline = **''**) as file:  
 writer=csv.writer(file)  
 writer.writerow([**'SY\_Sem'**, **'Population'**])  
 for i in range(0, len(semester)):  
 writer.writerow([str(semester[i]), str(number[i])])  
  
 *# Reading Students Dataset* population\_data=pd.read\_csv(**"Students Pop.csv"**)  
  
 *# Converting Month Column to DateTime* sem=population\_data[**'SY\_Sem'**]  
 pop=population\_data[**'Population'**]  
  
 *# Setting index of the Month* population\_data.set\_index(**'SY\_Sem'**, drop = True, inplace = True)  
  
 *# Spliting the dataset into train and test* train=population\_data[:-1]  
 test=population\_data[-1:]  
 adf\_test=ADFTest(alpha = 0.75)  
 pp\_test=PPTest(alpha = 0.58)  
  
 beta=adf\_test.should\_diff(train)[0] \* pp\_test.should\_diff(train)[0]  
  
 if beta < 0.01 or beta > 0.1:  
 diff=1  
 else:  
 diff=0  
  
 if adf\_test.should\_diff(train)[1] == True:  
 diff+=1  
 if pp\_test.should\_diff(train)[1] == True:  
 diff+=1  
 else:  
 diff-=1  
 DIFF=1  
 arima\_model=auto\_arima(train,  
 start\_p = 0, d = diff, start\_q = 0,  
 max\_p = 5, max\_d = 5, max\_q = 5,  
 start\_P = 0, D = DIFF, start\_Q = 0,  
 max\_P = 5, max\_D = 5, max\_Q = 5,  
 m = 3, seasonal = True,  
 information\_criterion = **'aic'**,  
 stepwise = True,  
 method = **'cg'**,  
 supress\_warnings = True, error\_action = **'warn'**, trace = True)  
  
 *#Model Summary* print(arima\_model.summary())  
  
 prediction=pd.DataFrame(arima\_model.predict(n\_periods = 1), index = test.index)  
 prediction.columns=[**'predicted\_population'**]  
 print(**'**\n**'** + str(round(prediction)))  
  
 *# Graphing all this* plt.figure(figsize = (16, 10))  
 plt.grid()  
 plt.plot(train, label = **"Training"**, marker = **'o'**)  
 plt.plot(test, label = **"Test"**, marker = **'o'**)  
 plt.plot(prediction, label = **"Predicted"**, marker = **'\*'**, linestyle = **'--'**)  
 plt.legend(loc = **'upper left'**)  
 plt.xticks(rotation = 45)  
 plt.figtext(0.22, 0.83,  
 str(round(prediction)),  
 verticalalignment = **"center"**,  
 wrap = True, fontsize = 13,  
 color = **"green"**)  
 plt.show()  
  
 *# Compute Accuracy* print(**'**\n**Prediction Accuracy Rate**\n**'**)  
 a=round(prediction.values.tolist()[0][0])  
 b=round(number[-1])  
 if b != 0:  
 print(str(round((100 - ((abs(b - a) \* 100) / b)), 1)) + **'%'**)  
 else:  
 print(**'The actual number is zero!'**)  
 Update()  
  
  
 def Results():  
 result=messagebox.askquestion(**"Save Results"**, **"Do you want to save results?"**, icon = **'question'**)  
 if result == **'yes'**:  
 SVR=Toplevel(root)  
 SVR.title(**"ECE-SPV"**)  
 SVR.wm\_iconbitmap(**"ece-spv.ico"**)  
 SVR.screen\_width=SVR.winfo\_screenwidth()  
 SVR.screen\_height=SVR.winfo\_screenheight()  
 SVR.geometry(**'%dx%d+%d+%d'** % (300, 160, (screen\_width / 2) - (width / 2), (screen\_height / 2) - (height / 2)))  
 SVR.resizable(0, 0)  
  
 def Confirm():  
 sys.stdout=restorePoint  
 if RESULT.get() == **""**:  
 txt\_result1.config(text = **"Please enter a folder name!"**, fg = **"red"**)  
 elif str(RESULT.get().split(**' '**, 1)[0]) == **"Trial"**:  
 result=messagebox.showwarning(**"Naming Warning"**, **"Please name the folder that start by 'Trial '!"**, icon = **'warning'**)  
 else:  
 dst=**"C:/Program Files/ECE-SPV/Result Saved"** outputresults=[**'arastdout.txt'**, **'xsection.xlsx'**, **'xinstructor.xlsx'**, **'xroom.xlsx'**, **'section.xlsx'**, **'instructor.xlsx'**, **'room.xlsx'**, **'settinginfo.txt'**]  
 dstfolder = os.listdir(dst)  
  
 def createFolder(dst):  
 try:  
 if not os.path.exists(dst):  
 os.makedirs(dst)  
 except OSError:  
 print(**'Error: Creating directory. '** + dst)  
 *# Creates a folder in the current directory called data* if str(RESULT.get()) not in dstfolder:  
 createFolder(dst + **'/'** + str(RESULT.get()) + **'/'**)  
 for i in range(0, len(dstfolder)):  
 if dstfolder[i][0:6] == **'Trial '**:  
 shutil.copytree(src = dst + **"/"** + str(dstfolder[i]),  
 dst = dst + **"/"** + str(RESULT.get()) + **"/"** + str(dstfolder[i]))  
 SVR.destroy()  
 txt\_result.config(text = **"The results are saved successfully."**, fg = **"green"**)  
 else:  
 txt\_result1.config(text = **"Destination folder already exists!"**, fg = **"red"**)  
  
 RESULT=StringVar(SVR)  
 frame=Frame(SVR, relief = **"raise"**)  
 frame.pack()  
 txt\_foldername=Label(frame, text = **"Enter a unique folder name:"**, font = (**'arial'**, 13), bd = 15)  
 txt\_foldername.pack()  
 e\_foldername=Entry(frame, textvariable = RESULT, width = 41)  
 e\_foldername.pack()  
 e\_foldername.delete(0, END)  
 Buttons=Frame(frame, bd = 8, relief = **"raise"**)  
 Buttons.pack(side=BOTTOM, pady=10)  
 txt\_result1=Label(Buttons)  
 txt\_result1.pack()  
 btn\_confirm=Button(Buttons, width = 28, text = **"Confirm"**, font = (**'arial'**, 10, **'bold'**),  
 fg = **"Black"**, command = lambda: Confirm())   
 btn\_confirm.pack()  
  
 txt\_result1.config(text = **"You can see folders at ../Result Saved/"**, fg = **"blue"**)  
  
 if \_\_name\_\_ == **'\_\_main\_\_'**:  
 SVR.mainloop()  
 else:  
 result=messagebox.askquestion(**"Save Results"**, **"Are you sure you don't want to save Results?"**, icon = **'question'**)  
 if result == **'yes'**:  
 try:  
 SVR.destroy()  
 except UnboundLocalError: pass  
 else: Results()  
  
  
 def Trial\_Results():  
 with open(SETTING\_INFO, **'r'**) as f:  
 def str2bool(v):  
 return v.lower() in (**"True"**, **"true"**, **"Yes"**, **"yes"**)  
  
 lines=f.readlines()  
 POPULATION\_SIZE=(int(lines[0].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 MUTATION\_RATE=(float(lines[1].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 NUMBER\_OF\_TRIALS=(int(lines[2].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 ALLOWABLE\_ROOM\_CONFLICTS=(float(lines[3].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 CONSECUTIVE\_TEACHING\_HOURS=(float(lines[4].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 SCHEDULING\_TIME\_LIMIT=(float(lines[5].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 RECURRING\_UPDATE\_TIME=(float(lines[6].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 VERBOSE\_FLAG=(str2bool(lines[7].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 ENABLE\_NUMB\_OF\_STUDENTS=(str2bool(lines[8].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 DISABLE\_CASUAL\_SPLITTING=(str2bool(lines[9].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 DISABLE\_UNEQUAL\_SPLIT=(str2bool(lines[10].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 DISABLE\_MIXED\_TYPE=(str2bool(lines[11].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 LAB\_ON\_VR=(str2bool(lines[12].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 DISABLE\_XDEPT\_ROOM\_UTILITY=(str2bool(lines[13].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 DISABLE\_INSTRUCTOR\_OVERLOAD=(str2bool(lines[14].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 LIMIT\_INSTRUCTOR\_AVAILABILITY=(str2bool(lines[15].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 LIMIT\_ROOM\_AVAILABILITY=(str2bool(lines[16].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 LIMIT\_SECTION\_AVAILABILITY=(str2bool(lines[17].split(**'='**, 1)[1].rstrip(**'**\n**'**)))  
 f.close()  
 sys.stdout=restorePoint  
 src=**"C:/Program Files/ECE-SPV"** dst=**"C:/Program Files/ECE-SPV/Result Saved"** outputresults=[**'arastdout.txt'**, **'xsection.xlsx'**, **'xinstructor.xlsx'**, **'xroom.xlsx'**, **'section.xlsx'**, **'instructor.xlsx'**, **'room.xlsx'**, **'settinginfo.txt'**]  
 dstfolder = os.listdir(dst)  
 srcfolder = os.listdir(src)  
 def createFolder(dst):  
 try:  
 if not os.path.exists(dst):  
 os.makedirs(dst)  
 except OSError:  
 print(**'Error: Creating directory. '** + dst)  
 *# Creates a folder in the current directory called data* for m in range(1, NUMBER\_OF\_TRIALS + 1):  
 if str(**"Trial "** + str(m)) not in dstfolder:  
 createFolder(dst + **'/'** + str(**"Trial "** + str(m)) + **'/'**)  
 for i in range(0, len(outputresults)):  
 if outputresults[i] in srcfolder:  
 shutil.copyfile(src = src + **"/"** + str(outputresults[i]),  
 dst = dst + **"/"** + str(**"Trial "** + str(m)) + **"/"** + str(outputresults[i]))  
 txt\_result.config(text = **"The trial result "** + str(m) + **" is temporarily saved."**, fg = **"green"**)  
 break  
 else: continue  
  
  
 def Refresh():  
 root.destroy()  
 ECESPV()  
  
  
 *# ==================================VARIABLES==========================================* DB=StringVar(root)  
 TEXTIN=StringVar(root)  
  
 *# ==================================FRAME==============================================* Top=Frame(root, width = 0, height = 0, bd = 8, relief = **"raise"**)  
 Top.pack(side = TOP)  
 Left=Frame(root, width = 300, height = 1000, bd = 8, relief = **"raise"**)  
 Left.pack(side = LEFT)  
 Right=Frame(root, width = 300, height = 1000, bd = 8, relief = **"raise"**)  
 Right.pack(side = RIGHT)  
 Forms=Frame(Left, width = 30, height = 10)  
 Forms.pack(side = BOTTOM)  
 TextOut=Frame(Right, width = 300, height = 10)  
 TextOut.pack(side = TOP)  
 Textin=Frame(Right, width = 300, height = 10)  
 Buttons=Frame(Left, width = 300, height = 450, bd = 8, relief = **"raise"**)  
 Buttons.pack(side = TOP)  
 txt\_title=Label(Top, width = 900, font = (**'arial'**, 24), text = **"DATABASE"**,  
 fg = **"Black"**)   
 txt\_title.pack()  
 txt\_result=Label(Buttons)  
 txt\_result.pack(side = TOP)  
  
 *# ==================================BUTTONS WIDGET=====================================* btn\_predict\_population=Button(Buttons, width = 35, text = **"Predict Population"**, command = threading.Thread(target=ARIMA).start)  
 btn\_predict\_population.pack(side = TOP)  
 btn\_run=Button(Buttons, width = 35, text = **"Show Initial Data"**, command = lambda: i())  
 btn\_run.pack(side = TOP)  
 btn\_find=Button(Buttons, width = 35, text = **"Find Schedule"**, command = threading.Thread(target=f).start)  
 btn\_find.pack(side = TOP)  
 cb\_db=ttk.Combobox(Buttons, textvariable = DB, width = 38)  
 cb\_db[**'values'**]=combobox\_input()  
 cb\_db.pack(side = TOP)  
 btn\_save=Button(Buttons, width = 35, text = **"SAVE DATABASE"**, command = Save)  
 btn\_save.pack(side = TOP)  
 btn\_load=Button(Buttons, width = 35, text = **"LOAD DATABASE"**, command = Load)  
 btn\_load.pack(side = TOP)  
 btn\_view=Button(Buttons, width = 35, text = **"VIEW DATABASE LIST"**, command = View)  
 btn\_delete=Button(Buttons, width = 35, text = **"DELETE DATABASE"**, command = Delete)  
 btn\_delete.pack(side = TOP)  
 btn\_add\_section=Button(Top, width = 12, text = **"Add Section"**, font = (**'arial'**, 10, **'bold'**), fg = **"Black"**,  
 bg = **"#CC99C9"**,  
 command = lambda: Add\_Section())   
 btn\_add\_section.pack(side = LEFT)  
 btn\_class\_creation=Button(Top, width = 12, text = **"Class Creation"**, font = (**'arial'**, 10, **'bold'**), fg = **"Black"**,  
 bg = **"#9EC1CF"**,  
 command = lambda: Class\_Creation())   
 btn\_class\_creation.pack(side = LEFT)  
 btn\_instructor=Button(Top, width = 12, text = **"Instructor"**, font = (**'arial'**, 10, **'bold'**), fg = **"Black"**,  
 bg = **"#9EE09E"**, command = lambda: Instructor())   
 btn\_instructor.pack(side = LEFT)  
 btn\_subject=Button(Top, width = 12, text = **"Subject"**, font = (**'arial'**, 10, **'bold'**), fg = **"Black"**, bg = **"#FDFD97"**,  
 command = lambda: Subject())   
 btn\_subject.pack(side = LEFT)  
 btn\_room=Button(Top, width = 12, text = **"Room"**, font = (**'arial'**, 10, **'bold'**), fg = **"Black"**, bg = **"#FEB144"**,  
 command = lambda: Room())   
 btn\_room.pack(side = LEFT)  
 btn\_population=Button(Top, width = 12, text = **"Population"**, font = (**'arial'**, 10, **'bold'**), fg = **"Black"**,  
 bg = **"#FF6663"**,  
 command = lambda: Population())   
 btn\_population.pack(side = LEFT)  
 btn\_refresh=Button(Top, width = 10, text = **"Refresh (F5)"**, font = (**'arial'**, 10, **'bold'**), fg = **"White"**, bg = **"Green"**,  
 command = Refresh)  
 btn\_refresh.pack(side = RIGHT)  
 btn\_outputs=Button(Top, width = 10, text = **"Output"**, font = (**'arial'**, 10, **'bold'**), fg = **"White"**, bg = **"#3639ff"**,  
 command = lambda: Outputs())  
 btn\_outputs.pack(side = RIGHT)  
 btn\_settings=Button(Top, width = 10, text = **"Settings"**, font = (**'arial'**, 10, **'bold'**), fg = **"White"**, bg = **"#c41e39"**,  
 command = lambda: Settings())  
 btn\_settings.pack(side = RIGHT)  
  
 *# ==================================LIST WIDGET========================================* scrollbary=Scrollbar(Forms, orient = VERTICAL)  
 scrollbarx=Scrollbar(Forms, orient = HORIZONTAL)  
 columns=(**"Number"**, **"Save Folder"**)  
 tree=ttk.Treeview(Forms, columns = columns, selectmode = **"extended"**, height = 22,  
 yscrollcommand = scrollbary.set, xscrollcommand = scrollbarx.set)   
  
  
 def treeview\_sort\_column(tree, col, reverse):  
 l=[(tree.set(k, col), k) for k in tree.get\_children(**''**)]  
 l.sort(reverse = reverse)  
  
 *# rearrange items in sorted positions* for index, (val, k) in enumerate(l):  
 tree.move(k, **''**, index)  
  
 *# reverse sort next time* tree.heading(col, command = lambda \_col=col: treeview\_sort\_column(tree, \_col, not reverse))  
  
  
 for col in columns:  
 tree.heading(col, text = col, command = lambda \_col=col: \  
 treeview\_sort\_column(tree, \_col, False))  
  
 *# for TREE Scroll Bars* scrollbary.config(command = tree.yview)  
 scrollbary.pack(side = RIGHT, fill = Y)  
 scrollbarx.config(command = tree.xview)  
 scrollbarx.pack(side = BOTTOM, fill = X)  
 *# inside the tree view* tree.heading(**'Number'**, text = **"No."**, anchor = W)  
 tree.heading(**'Save Folder'**, text = **"Save Folder"**, anchor = W)  
 tree.column(**'#0'**, stretch = NO, minwidth = 0, width = 0)  
 tree.column(**'#1'**, stretch = NO, minwidth = 40, width = 40)  
 tree.bind(**'<ButtonRelease-1>'**, click)  
 tree.pack()  
  
  
  
 *# ==================================TEXT OUT WIDGET========================================* textoutscrollbary=Scrollbar(TextOut, orient = VERTICAL)  
 textoutscrollbarx=Scrollbar(TextOut, orient = HORIZONTAL)  
 textout=tkinter.Text(TextOut, height = 60, width = 200, yscrollcommand = textoutscrollbary.set, xscrollcommand = textoutscrollbarx.set, wrap = **"none"**)   
 *# for TREE Scroll Bars* textoutscrollbary.config(command = textout.yview)  
 textoutscrollbary.pack(side = RIGHT, fill = Y)  
 textoutscrollbarx.config(command = textout.xview)  
 textoutscrollbarx.pack(side = BOTTOM, fill = X)  
  
 try:  
 with open(ARASTDOUT, **'r'**) as f:  
 textout.insert(END, f.read(), **'1.0'**)  
 f.close()  
 except IOError: *# this is what happens if the file doesn't exist* with open(ARASTDOUT, **'w'**) as f:  
 f.write(**''**)  
 f.close()  
  
 textout.pack()  
  
 *# ==================================TEXT IN WIDGET========================================* textin=Entry(Textin, textvariable = TEXTIN, width = 500)  
 textin.pack(side = BOTTOM)  
 textin.bind(**'<Return>'**, enter)  
  
 if \_\_name\_\_ == **'\_\_main\_\_'**:  
 Update()  
 app=FullScreenApp(root)  
 View()  
 root.mainloop()  
  
 *# Version 5 Clean*ECESPV()