from datetime import datetime  
from random import randint  
import hashlib  
  
def read\_all(file\_name):  
 read\_list = []  
 with open(file\_name, mode="r") as read\_file:  
 # Read each row from the particular file  
 for read\_line in read\_file:  
 # CONVERT the row to list  
 read\_line\_str = read\_line.replace("\n", "")  
 read\_line\_list = read\_line\_str.split(",")  
 # ADD the list to a master list  
 read\_list.append(read\_line\_list)  
 return read\_list  
  
  
def find\_one(file\_name, start\_reference\_column, end\_reference\_column, reference\_item):  
 # Read and retrieve one row that match the reference from the particular file  
 for read\_line in read\_all(file\_name):  
 find\_line = read\_line[start\_reference\_column: end\_reference\_column]  
 find\_line\_str = ",".join(find\_line)  
 if find\_line\_str == reference\_item:  
 return read\_line  
  
  
def find\_more(file\_name, start\_reference\_column, end\_reference\_column, reference\_item):  
 # Retrieve some row that match the reference from the particular file  
 find\_list = []  
 for read\_line in read\_all(file\_name):  
 find\_line = read\_line[start\_reference\_column: end\_reference\_column]  
 find\_line\_str = ",".join(find\_line)  
 if find\_line\_str == reference\_item:  
 find\_list.append(read\_line)  
 return find\_list  
  
  
def edit(file\_name, start\_reference\_column, end\_reference\_column, reference, edit\_column, edit\_item):  
 # Find the row that match the reference from the particular file  
 edit\_str = ""  
 for read\_line in read\_all(file\_name):  
 find\_line = read\_line[start\_reference\_column: end\_reference\_column]  
 find\_line\_str = ",".join(find\_line)  
 if find\_line\_str == reference:  
 # Update one information in the row  
 read\_line[edit\_column] = edit\_item  
 read\_line\_str = ",".join(read\_line)  
 edit\_str += read\_line\_str + "\n"  
 with open(file\_name, mode="w") as change\_file:  
 change\_file.write(edit\_str)  
  
  
def delete(file\_name, delete\_row):  
 print(delete\_row)  
 # Delete a row in the txt file  
 with open(file\_name, "r") as old\_file:  
 with open(file\_name, "r+") as new\_file:  
 start\_row = 0  
 while start\_row < delete\_row:  
 old\_file.readline()  
 start\_row += 1  
 current\_point = old\_file.tell()  
 new\_file.seek(current\_point, 0)  
 old\_file.readline()  
 next\_row = old\_file.readline()  
 while next\_row:  
 new\_file.write(next\_row)  
 next\_row = old\_file.readline()  
 old\_file.truncate()  
  
  
def register(file\_name, register\_list):  
 # Add a row with information to the txt file  
 register\_item = ",".join(register\_list) + "\n"  
 with open(file\_name, mode="a") as register\_file:  
 register\_file.write(register\_item)  
  
  
def retrieve\_level\_or\_subject(levels\_or\_subjects, level):  
 level\_and\_subject = {"FORM1": ["BAHASA MALAYSIA", "ENGLISH", "MATHS", "SCIENCE", "SEJARAH"],  
 "FORM2": ["BAHASA MALAYSIA", "ENGLISH", "MATHS", "SCIENCE", "SEJARAH"],  
 "FORM3": ["BAHASA MALAYSIA", "ENGLISH", "MATHS", "SCIENCE", "SEJARAH"],  
 "FORM4": ["BAHASA MALAYSIA", "ENGLISH", "MATHS", "SCIENCE", "SEJARAH", "ADD MATHS", "PHYSICS",  
 "CHEMISTRY", "BIOLOGY"],  
 "FORM5": ["BAHASA MALAYSIA", "ENGLISH", "MATHS", "SCIENCE", "SEJARAH", "ADD MATHS", "PHYSICS",  
 "CHEMISTRY", "BIOLOGY"]  
 }  
 if levels\_or\_subjects == "levels":  
 levels = level\_and\_subject.keys()  
 levels\_list = list(levels)  
 return levels\_list  
 else:  
 subjects\_in\_level = level\_and\_subject.get(level)  
 return subjects\_in\_level  
  
  
def retrieve\_subjects\_fee(subject\_name):  
 subjects\_fee = {"BAHASA MALAYSIA": 30,  
 "ENGLISH": 30,  
 "MATHS": 45,  
 "SCIENCE": 45,  
 "SEJARAH": 30,  
 "ADD MATHS": 50,  
 "ACCOUNTING": 45,  
 "ECONOMIC": 45,  
 "PHYSICS": 50,  
 "CHEMISTRY": 50,  
 "BIOLOGY": 50,  
 "NULL": 0  
 }  
 subject\_fee = subjects\_fee.get(subject\_name)  
 return subject\_fee  
  
  
def view\_levels\_and\_subjects():  
 levels\_list = retrieve\_level\_or\_subject("levels", None)  
 for level in levels\_list:  
 subjects\_list = retrieve\_level\_or\_subject("subjects", level)  
 subjects = " / ".join(subjects\_list)  
 print(f"{level}: {subjects}\n")  
  
  
def return\_home\_page(continue\_statement):  
 # Prompt user for return to home page  
 while True:  
 back\_choice = input(f"Enter 1 to return to home page / Enter 0 to {continue\_statement}: ")  
 if back\_choice == "1":  
 print("\nReturn to home page\n")  
 return True  
 elif back\_choice == "0":  
 return False  
 else:  
 print("Invalid command!!!\nPlease enter right command")  
  
  
def get\_and\_verify\_month():  
 # Get month and identify the validity of month  
 while True:  
 month = input("Please enter Month of the payment pay for (exp: January): ").capitalize().strip()  
 try:  
 datetime.strptime(month, "%B")  
 return month  
 except ValueError:  
 print("Invalid month")  
 if return\_home\_page("re-enter the month") is True:  
 return  
  
  
def get\_and\_verify\_year():  
 while True:  
 year = input("Please enter Year of the payment pay for (exp: 2023): ").strip()  
 try:  
 datetime.strptime(year, "%Y")  
 return year  
 except ValueError:  
 print("Invalid Year")  
 if return\_home\_page("re-enter the year") is True:  
 return  
  
  
def get\_and\_verify\_class\_date():  
 while True:  
 class\_date = input("Please enter Class Date (exp: 1 August 2023): ")  
 try:  
 datetime.strptime(class\_date, "%d %B %Y")  
 return class\_date  
 except ValueError:  
 print("Invalid date")  
 if return\_home\_page("re-enter class date") is True:  
 return  
  
  
def get\_and\_verify\_class\_venue():  
 while True:  
 class\_venue = input("Please enter Class Venue (exp: A-01): ").upper()  
 if class\_venue[0].isalpha() and class\_venue[1] == "-" and class\_venue[2:].isdigit():  
 return class\_venue  
 else:  
 print("Invalid Venue")  
 if return\_home\_page("re-enter class venue") is True:  
 return  
  
  
def get\_and\_verify\_class\_time(start\_or\_end):  
 while True:  
 class\_time = input(f"Please enter Class {start\_or\_end} Time (exp: 10.00 AM): ")  
 try:  
 datetime.strptime(class\_time, "%I.%M %p")  
 return class\_time  
 except ValueError:  
 print("Invalid Time")  
 if return\_home\_page(f"re-enter Class {start\_or\_end} Time") is True:  
 return  
  
  
def crypt\_password(user\_password):  
 sha256 = hashlib.sha256()  
 sha256.update(user\_password.encode("utf-8"))  
 user\_password\_sha256 = sha256.hexdigest()  
 return user\_password\_sha256  
  
  
def generate\_user\_id\_and\_password(role\_name):  
 if role\_name == "stud":  
 check\_file = "payment.txt"  
 else:  
 check\_file = role\_name + ".txt"  
 all\_user\_id = []  
 for user\_info in read\_all(check\_file):  
 all\_user\_id.append(user\_info[0])  
 while True:  
 code = str(randint(1, 9999))  
 id\_code = "0" \* (4 - len(code)) + code  
 user\_id = role\_name + id\_code  
 if user\_id not in all\_user\_id:  
 user\_password = user\_id + f"@B{id\_code}"  
 user\_password\_crypt = crypt\_password(user\_password)  
 return [user\_id, user\_password\_crypt]  
  
  
def get\_and\_verify\_role\_level\_for\_registration(prompt\_statement):  
 while True:  
 role\_level = input(f"Enter {prompt\_statement} to register the account (exp: FORM1): ").upper().strip()  
 levels\_list = retrieve\_level\_or\_subject("levels", None)  
 if role\_level in levels\_list:  
 return role\_level  
 else:  
 print("\nInvalid level")  
 if return\_home\_page(f"re-enter the {prompt\_statement}") is True:  
 return  
  
  
def get\_and\_verify\_stud\_id\_in\_payment\_file(prompt\_statement):  
 while True:  
 stud\_id = input(f"Please enter student id to {prompt\_statement}: ").strip()  
 stud\_info\_id = find\_one("stud.txt", 0, 1, stud\_id)  
 if stud\_info\_id:  
 return stud\_id  
 else:  
 print("Invalid Student ID")  
 if return\_home\_page("re-enter the student id") is True:  
 return  
  
  
def admin\_ui(user\_id):  
 # Admin Control Panel  
 admin\_functions = ["Register Receptionist", "Register Tutor", "View and Delete Receptionist",  
 "View and Delete Tutor", "View monthly income report", "Change Password", "Update Profile",  
 "Log out"]  
 # DECLARE admin\_function as temporarily storage  
 # CREATE a list which contain the function that admin can be used such as Register Receptionist and etc  
 # and store the list to admin\_function  
 while True:  
 print("-" \* 30 + "Admin Home Page" + "-" \* 30)  
 # DISPLAY Admin Home Page  
 # Display the function that can be used by admin  
 for index, function in enumerate(admin\_functions):  
 print(f"Enter {index + 1}: {function}")  
 # LOOP index and function from index 0 = element 1 to last index = last element step 1 of admin\_function  
 # CALCULATE index = index + 1  
 # DISPLAY index concatenate with function  
 # NEXT index and element  
 # ENDLOOP  
 # Prompt user for enter command  
 choice = input("Enter number: ")  
 if choice == "1":  
 register\_recep()  
 elif choice == "2":  
 register\_tutor()  
 elif choice == "3":  
 delete\_recep()  
 elif choice == "4":  
 delete\_tutor()  
 elif choice == "5":  
 view\_income()  
 elif choice == "6":  
 # CALL change\_password function where user\_file parameter is string(admin.txt) and user\_id parameter is user\_id  
 change\_password("admin.txt", user\_id)  
 elif choice == "7":  
 update\_profile("admin.txt", user\_id)  
 elif choice == "8":  
 print("Log out success\n")  
 return "out"  
 else:  
 print("Invalid command")  
  
  
def register\_recep():  
 # Register Receptionist  
 print("\nHome Page >>> Register Receptionist")  
 print("-" \* 30 + f"Register Receptionist" + "-" \* 30)  
 recep\_info\_list = ["Receptionist's Name",  
 "Receptionist's IC or Passport",  
 "Receptionist's Address",  
 "Receptionist's Contact Number",  
 "Receptionist's Email"]  
 new\_recep\_info = []  
 # Generate receptionist's id and password  
 id\_and\_password = generate\_user\_id\_and\_password("recep")  
 recep\_id = id\_and\_password[0]  
 print(f"Receptionist's ID generated: {recep\_id}")  
 recep\_password = id\_and\_password[1]  
 new\_recep\_info.append(recep\_id)  
 new\_recep\_info.append(recep\_password)  
 # Get receptionist's information and store into new recep information list  
 for recep\_data in recep\_info\_list:  
 data = input(f"Please enter {recep\_data} to register Receptionist account: ").strip()  
 new\_recep\_info.append(data)  
 register("recep.txt", new\_recep\_info)  
 print("-"\*30 + "\nRegister Receptionist success\n" + "-"\*30 + "\nReturn to home page\n")  
  
  
def register\_tutor():  
 # Register Tutor  
 print("\nHome Page >>> Register Tutor")  
 print("-" \* 30 + f"Register Tutor" + "-" \* 30)  
 tutor\_info\_list = ["Tutor's Name",  
 "Tutor's IC or Passport",  
 "Tutor's Address",  
 "Tutor's Contact Number",  
 "Tutor's Email"]  
 print("(Levels and Subjects)")  
 # View levels and subjects  
 view\_levels\_and\_subjects()  
 new\_tutor\_info = []  
 # Generate tutor's id and password  
 id\_and\_password = generate\_user\_id\_and\_password("tutor")  
 tutor\_id = id\_and\_password[0]  
 print(f"Tutor's ID generated: {tutor\_id}")  
 tutor\_password = id\_and\_password[1]  
 new\_tutor\_info.append(tutor\_id)  
 new\_tutor\_info.append(tutor\_password)  
 # Get tutor's information and store into new tutor information list  
 for tutor\_data in tutor\_info\_list:  
 data = input(f"Please enter {tutor\_data} to register Tutor account: ").strip()  
 new\_tutor\_info.append(data)  
 # Get tutor assign level and identify the validation of the tutor assign level  
 tutor\_assign\_level = get\_and\_verify\_role\_level\_for\_registration("Tutor's in-charge level")  
 if not tutor\_assign\_level:  
 return  
 else:  
 new\_tutor\_info.append(tutor\_assign\_level)  
 # Get tutor assign subject and identify the validation of the tutor assign subject  
 while True:  
 tutor\_assign\_subject = input(f"Enter Tutor's in-charge subject to register the account: ").upper().strip()  
 subject\_info = retrieve\_level\_or\_subject("subjects", tutor\_assign\_level)  
 # Identify the tutor subject whether exist or not  
 if tutor\_assign\_subject in subject\_info:  
 new\_tutor\_info.append(tutor\_assign\_subject)  
 break  
 else:  
 print("\nInvalid Subject")  
 if return\_home\_page(f"re-enter the Tutor's in-charge subject") is True:  
 return  
 register("tutor.txt", new\_tutor\_info)  
 print("-"\*30 + "\nRegister Tutor Success\n" + "-"\*30 + "\nReturn to home page\n")  
  
  
def delete\_recep():  
 print("\nHome Page >>> Delete Receptionist")  
 print("-" \* 30 + "Delete Receptionist" + "-" \* 30)  
 recep\_list = read\_all("recep.txt")  
 # DECLARE recep\_list as temporarily storage  
 # CALL read\_all function where file\_name parameter is string(recep.txt)  
 # recep\_list contain the return result of read\_all function  
 if not recep\_list:  
 # IF recep\_list is empty then  
 print("No Receptionist Information")  
 return  
 # Sort recep\_list order by the first element of recep\_list  
 recep\_list.sort(key=lambda id: (id[0], "recep"))  
 # View all receptionist information before deleting account  
 for recep\_info in recep\_list:  
 print(f"Receptionist ID: {recep\_info[0]}\nReceptionist Name: {recep\_info[2]}\n"  
 f"Receptionist IC or Passport: {recep\_info[3]}\nReceptionist Address: {recep\_info[4]}\n"  
 f"Receptionist Contact Number: {recep\_info[5]}\nReceptionist Email: {recep\_info[6]}\n"  
 + "~" \* 70)  
 # DISPLAY each element of recep\_info except the second element concatenate with its type  
 if return\_home\_page("delete receptionist's information") is True:  
 return  
 # Return None  
 # Get receptionist id and identify the validation of the receptionist id  
 while True:  
 delete\_recep\_id = input(f"Please enter Receptionist's ID to delete the account: ")  
 # PROMPT user to enter Receptionist's ID  
 # READ delete\_recep\_id  
 delete\_recep\_info = find\_one("recep.txt", 0, 1, delete\_recep\_id)  
 # DECLARE delete\_recep\_info as temporarily storage  
  
 # CALL find\_one function where file\_name parameter is string(recep.txt),  
 # start\_reference\_column parameter is integer(0), end\_reference\_column parameter is integer(1) ...  
  
 # delete\_recep\_info contain the return result of find\_one function  
  
 if delete\_recep\_info:  
 # IF delete\_recep\_info is not empty then  
 recep\_delete\_row = recep\_list.index(delete\_recep\_info)  
 # DECLARE recep\_delete\_row as temporarily storage  
 # FIND the index value of delete\_recep\_info in recep\_list and store the index to recep\_delete\_row  
 break  
 else:  
 print("\nReceptionist ID does not exist")  
 if return\_home\_page("re-enter the receptionist's id") is True:  
 return  
 delete("recep.txt", recep\_delete\_row)  
 print("-"\*30 + "\nDelete Receptionist Success\n" + "-"\*30 + "\nReturn to home page\n")  
  
  
def delete\_tutor():  
 print(f"\nHome Page >>> Delete Tutor")  
 print("-" \* 30 + f"Delete Receptionist" + "-" \* 30)  
 tutor\_list = read\_all("tutor.txt")  
 if not tutor\_list:  
 print("No Tutor Information")  
 return  
 tutor\_list.sort(key=lambda id: (id[0], "tutor"))  
 # Allow admin to view all tutor information before deleting account  
 for tutor\_info in tutor\_list:  
 print(f"Tutor's ID: {tutor\_info[0]}\nTutor's Name: {tutor\_info[2]}\n"  
 f"Tutor's IC or Passport: {tutor\_info[3]}\nTutor's Address: {tutor\_info[4]}\n"  
 f"Tutor's Contact Number: {tutor\_info[5]}\nTutor's Email: {tutor\_info[6]}\n"  
 f"Tutor's Assign Level: {tutor\_info[7]}\nTutor's Assign Subject: {tutor\_info[8]}\n"  
 + "~" \* 70)  
 if return\_home\_page("delete tutor's information") is True:  
 return  
 # Get tutor id and identify the validation of the tutor id for deleting account  
 while True:  
 delete\_tutor\_id = input(f"Please enter Tutor ID to delete the account: ").strip()  
 delete\_tutor\_info = find\_one("tutor.txt", 0, 1, delete\_tutor\_id)  
 if delete\_tutor\_info:  
 tutor\_delete\_row = tutor\_list.index(delete\_tutor\_info)  
 break  
 else:  
 print("\nTutor's ID does not exist")  
 if return\_home\_page("re-enter the tutor's id") is True:  
 return  
 delete("tutor.txt", tutor\_delete\_row)  
 print("-"\*30 + "\nDelete Tutor Success\n" + "-"\*30 + "\nReturn to home page\n")  
  
  
def view\_income():  
 print("\nHome Page >>> View Student Monthly Income")  
 print("-"\*30 + "View Monthly Income" + "-"\*30)  
 # Get month and identify the validation of month  
 # Get year and identify the validation of year  
 month = get\_and\_verify\_month()  
 # DECLARE month as temporarily storage  
 # CALL get\_and\_verify\_month function  
 # month contain the return result of get\_and\_verify\_month function  
 year = get\_and\_verify\_year()  
 if not month or not year:  
 return  
 date = f"{month} {year}"  
 # Retrieve student payment which in required date, level and payment status  
 payment\_list\_date = find\_more("payment.txt", 1, 2, date)  
 payment\_list\_status = find\_more("payment.txt", 10, 11, "PAID")  
 payment\_list\_date\_level\_status = []  
 if not payment\_list\_status or not payment\_list\_date:  
 print("No available of income in this month\nReturn to home page\n")  
 return  
 else:  
 for payment\_info\_status in payment\_list\_status:  
 if payment\_info\_status in payment\_list\_date:  
 payment\_list\_date\_level\_status.append(payment\_info\_status)  
 print(f"\nDate: {date}\n")  
 # Calculate the total income of each subject and calculate the total income in the month  
 total\_fee = 0  
 # Retrieve each subject in the level  
 for level in retrieve\_level\_or\_subject("levels", None):  
 total\_level\_fee = 0  
 print(f"Level: {level}")  
 subject\_info = retrieve\_level\_or\_subject("subjects", level)  
 for subject\_num, subject in enumerate(subject\_info):  
 total\_subject\_fee = 0  
 # Loop 1: Find student payment list which subject 1 is the subject  
 # Loop 2: Find student payment list which subject 2 is the subject  
 # Loop 3: Find student payment list which subject 3 is the subject  
 for subject\_enrolled\_num in range(3):  
 payment\_list\_subject\_enrolled = find\_more("payment.txt", 4 + subject\_enrolled\_num, 5 + subject\_enrolled\_num, subject)  
 if payment\_list\_subject\_enrolled:  
 payment\_list\_level = find\_more("payment.txt", 2, 3, level)  
 for payment\_info\_subject\_enrolled in payment\_list\_subject\_enrolled:  
 if payment\_info\_subject\_enrolled in payment\_list\_date\_level\_status and payment\_info\_subject\_enrolled in payment\_list\_level:  
 subject\_fee = payment\_info\_subject\_enrolled[subject\_enrolled\_num + 7]  
 total\_subject\_fee += int(subject\_fee)  
 # CONVERT subject\_fee to integer  
 # total\_subject\_fee = total\_subject\_fee + subject\_fee  
 print(f"\tSubject{subject\_num + 1}: {subject}\n\tTotal income: RM{total\_subject\_fee}\n")  
 total\_level\_fee += total\_subject\_fee  
 total\_fee += total\_level\_fee  
 print(f"Total income of {level}: {total\_level\_fee}\n")  
 print("-" \* 30 + f"\nTotal income of {month} {year} is RM {total\_fee}\n" + "-" \* 30 + "\nReturn to home page")  
  
  
def recep\_ui(user\_id):  
 recep\_functions = ["Register and Enroll Students", "View Student's Request and Update student subject enrollment",  
 "Generate Student Payment", "Accept payment", "Generate receipt", "View and Delete Student",  
 "Add News", "Delete News", "Change Password", "Update Profile", "Log out"]  
 while True:  
 print("-" \* 30 + "Receptionist Home Page" + "-" \* 30)  
 for index, function in enumerate(recep\_functions):  
 print(f"Enter {index + 1}: {function}")  
 choice = input("Enter number: ")  
 if choice == "1":  
 register\_stud()  
 elif choice == "2":  
 update\_stud()  
 elif choice == "3":  
 generate\_stud\_payment()  
 elif choice == "4":  
 accept\_payment(user\_id)  
 elif choice == "5":  
 generate\_receipt()  
 elif choice == "6":  
 delete\_stud()  
 elif choice == "7":  
 add\_news()  
 elif choice == "8":  
 delete\_news()  
 elif choice == "9":  
 change\_password("recep.txt", user\_id)  
 elif choice == "10":  
 update\_profile("recep.txt", user\_id)  
 elif choice == "11":  
 print("Log out success")  
 return "out"  
 else:  
 print("Invalid command")  
  
  
def register\_stud():  
 print("\nHome Page >>> Register Students")  
 print("-" \* 30 + "Register Students" + "-" \* 30)  
 stud\_info\_list = ["Student's Name",  
 "Student's IC or Passport",  
 "Student's Address",  
 "Student's Contact Number",  
 "Student's Email"]  
 # View levels and subjects  
 view\_levels\_and\_subjects()  
 new\_stud\_info = []  
 # Generate student's id and password  
 id\_and\_password = generate\_user\_id\_and\_password("stud")  
 stud\_id = id\_and\_password[0]  
 print(f"Student's ID generated: {stud\_id}")  
 stud\_password = id\_and\_password[1]  
 new\_stud\_info.append(stud\_id)  
 new\_stud\_info.append(stud\_password)  
 # Get student's information and store into new stud information list  
 for stud\_data in stud\_info\_list:  
 data = input(f"Please enter {stud\_data} to register Student account: ").strip()  
 new\_stud\_info.append(data)  
 month\_of\_enrollment = datetime.now().strftime("%d %B %Y")  
 new\_stud\_info.append(month\_of\_enrollment)  
 # Get student level and identify the validity of the level  
 stud\_level = get\_and\_verify\_role\_level\_for\_registration("Student's level")  
 if not stud\_level:  
 return  
 else:  
 new\_stud\_info.append(stud\_level)  
 # Get student subject 1, subject2 and subject3 and identify the validity of the subject  
 for stud\_subject\_num in range(1, 4):  
 while True:  
 student\_subject = input(  
 f"Enter Student Subject {stud\_subject\_num} to register the account (Enter NULL if no subject enroll): ").upper().strip()  
 subject\_info = retrieve\_level\_or\_subject("subjects", stud\_level)  
 # Identify the tutor subject whether exist or not  
 if student\_subject in subject\_info or student\_subject == "NULL":  
 new\_stud\_info.append(student\_subject)  
 break  
 else:  
 print("\nInvalid Subject")  
 if return\_home\_page(f"re-enter the Subject {stud\_subject\_num}") is True:  
 return  
 register("stud.txt", new\_stud\_info)  
 generate\_payment\_after\_register(stud\_id)  
 print("-"\*30 + "\nRegister Student Success\nAdd Student Education fee list of this month Success\n" + "-"\*30 +  
 "\nReturn to home page\n")  
  
  
def generate\_payment\_after\_register(stud\_id):  
 stud\_info\_id = find\_one("stud.txt", 0, 1, stud\_id)  
 date\_of\_education\_fee = datetime.now().strftime("%B %Y")  
 date\_of\_payment\_done = "NULL"  
 level = stud\_info\_id[8]  
 subject1 = stud\_info\_id[9]  
 subject2 = stud\_info\_id[10]  
 subject3 = stud\_info\_id[11]  
 subject1\_fee = str(retrieve\_subjects\_fee(subject1))  
 subject2\_fee = str(retrieve\_subjects\_fee(subject2))  
 subject3\_fee = str(retrieve\_subjects\_fee(subject3))  
 payment\_status = "UNPAID"  
 accept\_payment\_by = "NULL"  
 new\_payment\_info = [stud\_id, date\_of\_education\_fee, level, date\_of\_payment\_done, subject1, subject2,  
 subject3, subject1\_fee, subject2\_fee, subject3\_fee, payment\_status, accept\_payment\_by]  
 register("payment.txt", new\_payment\_info)  
  
  
def update\_stud():  
 print("\nHome Page >>> View Student's Request and Update Student Enrollment Subject")  
 print("-" \* 30 + "View Student's Request and Update Student Enrollment Subject" + "-" \* 30)  
 if not view\_stud\_request():  
 return  
 if return\_home\_page("update student information") is True:  
 return  
 # Get student id and identify the validation of the student id  
 while True:  
 request\_no = input("Enter the Request No. to approve or reject the request: ").strip()  
 if request\_no.isdigit():  
 request\_info\_request\_no = find\_one("request.txt", 0, 1, request\_no)  
 if request\_info\_request\_no:  
 break  
 print("Invalid Request No.")  
 if return\_home\_page("re-enter the request no.") is True:  
 return  
 stud\_id = request\_info\_request\_no[1]  
 change\_subject = request\_info\_request\_no[2]  
 new\_subject = request\_info\_request\_no[3]  
 stud\_info\_id = find\_one("stud.txt", 0, 1, stud\_id)  
 change\_subject\_column = stud\_info\_id.index(change\_subject)  
 while True:  
 approve\_reject = input("Enter APPROVE to approve the request / Enter REJECT to reject the request: ").strip().upper()  
 if approve\_reject in ["APPROVE", "REJECT"]:  
 break  
 else:  
 print("You can only enter APPROVE or REJECT")  
 if return\_home\_page("re-enter APPROVE or REJECT") is True:  
 return  
 if approve\_reject == "APPROVE":  
 edit("stud.txt", 0, 1, stud\_id, change\_subject\_column, new\_subject)  
 edit("request.txt", 0, 1, request\_no, 5, "APPROVED")  
 print("-"\*30 + "\nStudent's Request approved and Update Student Enrollment Subject Success\n" + "-"\*30 +  
 "\nReturn to home page\n")  
 else:  
 edit("request.txt", 0, 1, request\_no, 5, "REJECTED")  
 print("-"\*30 + "\nStudent's Request rejected\n" + "-"\*30 + "\nReturn to home page\n")  
  
  
def view\_stud\_request():  
 stud\_request\_list\_pending = find\_more("request.txt", 5, None, "STILL PENDING")  
 if not stud\_request\_list\_pending:  
 print("No available of student request in still pending\nReturn to home page\n")  
 return  
 # View student payment list which in still pending status  
 for stud\_request\_info\_pending in stud\_request\_list\_pending:  
 print(f"Request No.: {stud\_request\_info\_pending[0]}\nStudent ID: {stud\_request\_info\_pending[1]}\n"  
 f"Change subject enrollment from {stud\_request\_info\_pending[2]} to "  
 f"{stud\_request\_info\_pending[3]}\nReason: {stud\_request\_info\_pending[4]}\n" + "~" \* 70)  
 return 1  
  
  
def generate\_stud\_payment():  
 print("\nHome Page >>> Generate Student Payment")  
 print("-" \* 30 + "Generate Student Payment" + "-" \* 30)  
 if return\_home\_page("generate all student education fee list in this month") is True:  
 return  
 # Get student id and identify the validation of the student id  
 stud\_list = read\_all("stud.txt")  
 date\_of\_generate\_education\_fee = datetime.now().strftime("%B %Y")  
 for stud\_info in stud\_list:  
 stud\_id = stud\_info[0]  
 stud\_level = stud\_info[8]  
 payment\_date = "NULL"  
 subject1 = stud\_info[9]  
 subject2 = stud\_info[9]  
 subject3 = stud\_info[10]  
 subject1\_fee = str(retrieve\_subjects\_fee(subject1))  
 subject2\_fee = str(retrieve\_subjects\_fee(subject2))  
 subject3\_fee = str(retrieve\_subjects\_fee(subject3))  
 payment\_status = "UNPAID"  
 accept\_payment\_by = "NULL"  
 new\_payment\_info = [stud\_id, date\_of\_generate\_education\_fee, stud\_level, payment\_date, subject1, subject2,  
 subject3, subject1\_fee, subject2\_fee, subject3\_fee, payment\_status, accept\_payment\_by]  
 register("payment.txt", new\_payment\_info)  
 print("-"\*30 + "\nGenerate Student Payment Success\n" + "-"\*30 + "\nReturn to home page\n")  
  
  
def view\_student\_payment\_in\_unpaid():  
 stud\_payment\_list\_unpaid = find\_more("payment.txt", 10, 11, "UNPAID")  
 if not stud\_payment\_list\_unpaid:  
 print("Unavailable of student unpaid payment")  
 return  
 stud\_payment\_list\_unpaid.sort(key=lambda id: (id[0], ""))  
 # View all student payment information which in unpaid status  
 for stud\_payment\_info\_unpaid in stud\_payment\_list\_unpaid:  
 print(f"Student ID: {stud\_payment\_info\_unpaid[0]}\nMonth: {stud\_payment\_info\_unpaid[1]}\n"  
 f"Level: {stud\_payment\_info\_unpaid[2]}\nPayment Detail: "  
 f"Subject 1 {stud\_payment\_info\_unpaid[4]} (RM{stud\_payment\_info\_unpaid[7]}) | "  
 f"Subject 2 {stud\_payment\_info\_unpaid[5]} (RM{stud\_payment\_info\_unpaid[8]}) | "  
 f"Subject 3 {stud\_payment\_info\_unpaid[6]} (RM{stud\_payment\_info\_unpaid[9]})\n"  
 f"Payment Status: {stud\_payment\_info\_unpaid[10]}\n" + "~" \* 100)  
 return stud\_payment\_list\_unpaid  
  
  
def accept\_payment(user\_id):  
 print("\nHome Page >>> Accept payment")  
 print("-" \* 30 + "Accept Payment" + "-" \* 30)  
 # Retrieve all student payment list which in unpaid status  
 stud\_payment\_list\_unpaid = view\_student\_payment\_in\_unpaid()  
 if not stud\_payment\_list\_unpaid:  
 return  
 if return\_home\_page("accept payment") is True:  
 return  
 # Get student id and identity the validity of student id for accept payment  
 stud\_id = get\_and\_verify\_stud\_id\_in\_payment\_file("accept the payment")  
 if not stud\_id:  
 return  
 # Get month and identify the validity of month  
 # Get year and identify the validity of year  
 month = get\_and\_verify\_month()  
 year = get\_and\_verify\_year()  
 if not month or not year:  
 return  
 date = f"{month} {year}"  
 stud\_payment\_list\_id = find\_more("payment.txt", 0, 1, stud\_id)  
 stud\_payment\_list\_date = find\_more("payment.txt", 1, 2, date)  
 # Retrieve student payment information which in required student id, date and status  
 stud\_payment\_list\_id\_date\_unpaid = []  
 for stud\_payment\_info\_unpaid in stud\_payment\_list\_unpaid:  
 if stud\_payment\_info\_unpaid in stud\_payment\_list\_date and stud\_payment\_info\_unpaid in stud\_payment\_list\_id:  
 stud\_payment\_list\_id\_date\_unpaid = stud\_payment\_info\_unpaid  
 if not stud\_payment\_list\_id\_date\_unpaid:  
 print("Unavailable of student payment which contain required ID, Month and Year")  
 return  
 # Get the payment date for record  
 else:  
 payment\_date = datetime.now().strftime("%d %B %Y")  
 recep\_name = find\_one("recep.txt", 0, 1, user\_id)[2]  
 edit("payment.txt", 0, 2, f"{stud\_id},{date}", 11, recep\_name)  
 edit("payment.txt", 0, 2, f"{stud\_id},{date}", 3, payment\_date)  
 edit("payment.txt", 0, 2, f"{stud\_id},{date}", 10, "PAID")  
 print("-"\*30 + "\nAccept Payment Success\n" + "-"\*30 + "\nReturn to home page\n")  
  
  
def generate\_receipt():  
 print("\nHome Page >>> Generate Receipt")  
 print("-" \* 30 + "Generate Receipt" + "-" \* 30)  
 # Retrieve all student payment information which in paid status  
 stud\_payment\_list\_paid = find\_more("payment.txt", 10, 11, "PAID")  
 if not stud\_payment\_list\_paid:  
 print("No available of Student Payment which in paid status\nReturn to home page\n")  
 return  
 # Get student id and identify the validity of the student id for generating receipt  
 stud\_id = get\_and\_verify\_stud\_id\_in\_payment\_file("generate the receipt")  
 if not stud\_id:  
 return  
 # Get month and identify the validity of month  
 # Get year and identify the validity of year  
 month = get\_and\_verify\_month()  
 year = get\_and\_verify\_year()  
 if month is None or year is None:  
 return  
 date = f"{month} {year}"  
 stud\_payment\_list\_id = find\_more("payment.txt", 0, 1, stud\_id)  
 stud\_payment\_list\_date = find\_more("payment.txt", 1, 2, date)  
 # Retrieve student payment information which in required student id, date and status  
 stud\_payment\_list\_id\_date\_paid = []  
 for stud\_payment\_info\_paid in stud\_payment\_list\_paid:  
 if stud\_payment\_info\_paid in stud\_payment\_list\_date and stud\_payment\_info\_paid in stud\_payment\_list\_id:  
 stud\_payment\_list\_id\_date\_paid = stud\_payment\_info\_paid  
 if not stud\_payment\_list\_id\_date\_paid:  
 print("\nNo available of student payment which contain required ID, Month and Year and Status\nReturn to home page\n")  
 return  
 else:  
 total\_amount\_paid = 0  
 for amount\_paid in stud\_payment\_list\_id\_date\_paid[7:10]:  
 total\_amount\_paid += int(amount\_paid)  
 # Generate Receipt  
 stud\_info\_id = find\_one("stud.txt", 0, 1, stud\_id)  
 print("~" \* 30 + "Student Receipt" + "~" \* 30)  
 print(f"Student ID: {stud\_payment\_list\_id\_date\_paid[0]}\nStudent Name: {stud\_info\_id[2]}\n"  
 f"Month of Education fee: {stud\_payment\_list\_id\_date\_paid[1]}\n"  
 f"Payment Date: {stud\_payment\_list\_id\_date\_paid[3]}\nTotal Amount: RM{total\_amount\_paid}"  
 f"\nAccept payment by Receptionist: {stud\_payment\_list\_id\_date\_paid[11]}\n")  
 print("Student's Receipt Generated\nReturn to home page\n")  
  
  
def delete\_stud():  
 print("\nHome Page >>> Delete Student")  
 print("-" \* 30 + "Delete Student" + "-" \* 30)  
 stud\_info\_list = ["Student's ID", "Student's Password", "Student's Name", "Student's IC or Passport",  
 "Student's Address", "Student's Email", "Student's Contact Number", "Month of Enrollment"]  
 # Retrieve student information  
 stud\_list = read\_all("stud.txt")  
 if not stud\_list:  
 print("No student information")  
 return  
 stud\_list.sort(key=lambda id: (id[0], "stud"))  
 # View all student information  
 for stud\_info in stud\_list:  
 for index, stud\_data in enumerate(stud\_info[0:8]):  
 if index != 1:  
 print(f"{stud\_info\_list[index]}: {stud\_data}")  
 print("~"\*70)  
 if return\_home\_page("delete student's account") is True:  
 return  
 # Get student id and identify the validation of the student id  
 while True:  
 stud\_id = input("Please enter student id to delete the account: ")  
 stud\_info\_id = find\_one("stud.txt", 0, 1, stud\_id)  
 if stud\_info\_id:  
 # find the index of the student information in the student list and delete it  
 delete\_row = stud\_list.index(stud\_info\_id)  
 break  
 else:  
 print("This student id does not exist")  
 if return\_home\_page("re-enter the student's id") is True:  
 return  
 delete("stud.txt", delete\_row)  
 print("-" \* 30 + "\nDelete Student Information Success\n" + "-" \* 30 + "\nReturn to home page\n")  
  
  
def add\_news():  
 print("\nHome Page >>> Add News")  
 print("-" \* 30 + "Add News" + "-" \* 30)  
 new\_news\_str = ""  
 num = 1  
 print("You can add your news (Enter NULL if no more news)")  
 while True:  
 news\_data = input(f"Paragraph {num}: ")  
 if news\_data.upper() == "NULL":  
 break  
 else:  
 new\_news\_str += news\_data + "\\n"  
 num += 1  
 new\_news\_list = [new\_news\_str]  
 register("news.txt", new\_news\_list)  
 print("-"\*30 + "\nAdd News Success\n" + "-"\*30 + "\nReturn to home page\n")  
  
  
def delete\_news():  
 print("\nHome Page >>> Delete News")  
 print("-" \* 30 + "Delete News" + "-" \* 30)  
 if not view\_news():  
 print("No any news")  
 return  
 if return\_home\_page("delete news") is True:  
 return  
 while True:  
 delete\_news\_num = input("Please enter the News Number that you would like to delete: ")  
 try:  
 delete\_news\_num = int(delete\_news\_num)  
 if 0 < delete\_news\_num <= len(read\_all("news.txt")):  
 break  
 else:  
 print("Invalid news no")  
 except ValueError:  
 print("Invalid news no")  
 if return\_home\_page("re-enter the News Number") is True:  
 return  
 delete("news.txt", delete\_news\_num-1)  
 print("-"\*30 + "\nDelete News Success\n" + "-"\*30 + "\nReturn to home page\n")  
  
  
def tutor\_ui(user\_id):  
 tutor\_functions = ["Add class information", "Update class information", "Delete Class Information",  
 "View Student List in the Subject", "Change Password", "Update Profile", "Log out"]  
 while True:  
 print("-" \* 30 + "Tutor Home Page" + "-" \* 30)  
 for index, function in enumerate(tutor\_functions):  
 print(f"Enter {index + 1}: {function}")  
 choice = input("Enter number: ")  
 if choice == "1":  
 add\_class(user\_id)  
 elif choice == "2":  
 update\_class(user\_id)  
 elif choice == "3":  
 delete\_class(user\_id)  
 elif choice == "4":  
 view\_stud(user\_id)  
 elif choice == "5":  
 change\_password("tutor.txt", user\_id)  
 elif choice == "6":  
 update\_profile("tutor.txt", user\_id)  
 elif choice == "7":  
 print("Log out success")  
 return "out"  
 else:  
 print("Invalid command")  
  
  
def add\_class(user\_id):  
 print("\nHome Page >>> Add Class Schedule")  
 print("-" \* 30 + "Add Class Schedule" + "-" \* 30)  
 # Retrieve tutor name, assign level and assign subject  
 tutor\_info = find\_one("tutor.txt", 0, 1, user\_id)  
 tutor\_name = tutor\_info[2]  
 tutor\_assign\_level = tutor\_info[7]  
 tutor\_assign\_subject = tutor\_info[8]  
 subject\_charge = str(retrieve\_subjects\_fee(tutor\_assign\_subject))  
 class\_date = get\_and\_verify\_class\_date()  
 if not class\_date:  
 return  
 class\_venue = get\_and\_verify\_class\_venue()  
 if not class\_venue:  
 return  
 class\_start\_time = get\_and\_verify\_class\_time("Start")  
 if not class\_start\_time:  
 return  
 class\_end\_time = get\_and\_verify\_class\_time("End")  
 if not class\_end\_time:  
 return  
 new\_class\_info = [class\_date, class\_venue, class\_start\_time, class\_end\_time, tutor\_assign\_level,  
 tutor\_assign\_subject, subject\_charge, tutor\_name, user\_id]  
 register("class.txt", new\_class\_info)  
 print("-" \* 30 + "\nClass information has been saved\n" + "-" \* 30 + "\nReturn to home page\n")  
  
  
def tutor\_view\_class(user\_id):  
 # Create class information list to store all needed class information  
 class\_info\_list = ["Date", "Venue", "Start time", "End time", "Level", "Subject", "Charge", "Tutor name", "Tutor id"]  
 # Retrieve tutor class schedule and sort by date and start time  
 tutor\_class\_list = find\_more("class.txt", 8, None, user\_id)  
 if not tutor\_class\_list:  
 print("No class schedule")  
 return  
 tutor\_class\_list.sort(key=lambda date: (datetime.strptime(date[0], "%d %B %Y"), datetime.strptime(date[2], "%H.%M %p")))  
 for index, tutor\_class in enumerate(tutor\_class\_list):  
 tutor\_class\_str = ""  
 tutor\_class\_str += f"(Class No. {index + 1})"  
 for class\_data\_num, class\_data in enumerate(class\_info\_list):  
 tutor\_class\_str += f"\n{class\_data}: {tutor\_class[class\_data\_num]}"  
 tutor\_class\_str += "\n" + "~" \* 70  
 print(tutor\_class\_str)  
 return tutor\_class\_list  
  
  
def update\_class(user\_id):  
 print("\nHome Page >>> Update Class Information")  
 print("-" \* 30 + "Update Class Information" + "-" \* 30)  
 # View all class schedule created by the tutor  
 class\_list = tutor\_view\_class(user\_id)  
 class\_info\_list = ["Date", "Venue", "Start time", "End time", "Level", "Subject", "Charge", "Tutor name", "Tutor id"]  
 if not class\_list:  
 return  
 # Tutor can decide return to home page when don't want update class information after viewing the class schedule  
 if return\_home\_page("update class schedule") is True:  
 return  
 # Get class no. and identity the validation of the class no.  
 while True:  
 update\_choice = input("Please enter Class No. to update the class information: ")  
 try:  
 update\_choice = int(update\_choice)  
 if 0 < update\_choice <= len(class\_list):  
 chosen\_class = class\_list[update\_choice - 1]  
 break  
 else:  
 print("\nClass No. does not exist!!!")  
 if return\_home\_page("re-enter the class no.") is True:  
 return  
 except ValueError:  
 print("\nClass No. does not exist!!!")  
 if return\_home\_page("re-enter the class no.") is True:  
 return  
 # Get class information column and identity the validation of the class information column  
 while True:  
 class\_data = input("Please enter one class information to edit the information (exp: Date): ").capitalize()  
 if class\_data in class\_info\_list[:4]:  
 class\_column = class\_info\_list.index(class\_data)  
 break  
 elif class\_data in class\_info\_list[4:]:  
 print("\nYou cannot update this information\nPlease enter again")  
 else:  
 print("Invalid class information!!!")  
 if return\_home\_page("re-enter the class information") is True:  
 return  
 if class\_data == "Date":  
 new\_data = get\_and\_verify\_class\_date()  
 elif class\_data == "Venue":  
 new\_data = get\_and\_verify\_class\_venue()  
 elif class\_data == "Start Time":  
 new\_data = get\_and\_verify\_class\_time("Start")  
 else:  
 new\_data = get\_and\_verify\_class\_time("End")  
 if not new\_data:  
 return  
 chosen\_class\_str = ",".join(chosen\_class)  
 edit("class.txt", 0, None, chosen\_class\_str, class\_column, new\_data)  
 print("-" \* 30 + "\nUpdate Class success\n" + "-" \* 30 + "\nReturn to home page\n")  
  
  
def delete\_class(user\_id):  
 print("\nHome Page >>> Delete Class Information")  
 print("-" \* 30 + "Delete Class Information" + "-" \* 30)  
 # View all class schedule created by the tutor  
 class\_list = tutor\_view\_class(user\_id)  
 if not class\_list:  
 return  
 # Tutor can decide return to home page when don't want to delete class information after viewing the class schedule  
 if return\_home\_page("delete class schedule") is True:  
 return  
 while True:  
 # Prompt user for choose the class that would like to update  
 delete\_choice = input("Please enter Class No. to delete the class information: ")  
 try:  
 delete\_choice = int(delete\_choice)  
 # Identity the validity of class no.  
 if 0 < delete\_choice <= len(class\_list):  
 break  
 else:  
 print("\nClass No. does not exist!!!")  
 if return\_home\_page("re-enter the class no.") is True:  
 return  
 except ValueError:  
 print("\nClass No. does not exist!!!")  
 if return\_home\_page("re-enter the class no.") is True:  
 return  
 chosen\_class = class\_list[delete\_choice - 1]  
 for class\_row, class\_list in enumerate(read\_all("class.txt")):  
 if chosen\_class == class\_list:  
 delete\_row = class\_row  
 delete("class.txt", delete\_row)  
 print("-"\*30 + "\nDelete Class Information Success\n" + "-"\*30 + "\nReturn to home page\n")  
 break  
  
  
def view\_stud(user\_id):  
 print("\nHome Page >>> View Student List in the Subject")  
 print("-" \* 30 + "View Student List in the Subject" + "-" \* 30)  
 print("-" \* 8 + "(Student List)" + "-" \* 8)  
 # Get tutor assign level and subject  
 tutor\_info = find\_one("tutor.txt", 0, 1, user\_id)  
 assign\_level = tutor\_info[7]  
 assign\_subject = tutor\_info[8]  
 stud\_list\_assign\_level = find\_more("stud.txt", 8, 9, assign\_level)  
 stud\_list = read\_all("stud.txt")  
 stud\_list\_assign\_subject = []  
 for stud\_information in stud\_list:  
 if assign\_subject in stud\_information[9:]:  
 stud\_list\_assign\_subject.append(stud\_information)  
 if not stud\_list\_assign\_level or not stud\_list\_assign\_subject:  
 print("No student in this class\n" + "\nReturn to home page\n")  
 return  
 for stud\_info\_assign\_subject in stud\_list\_assign\_subject:  
 if stud\_info\_assign\_subject in stud\_list\_assign\_level:  
 print(f"Student ID: {stud\_info\_assign\_subject[0]} | Student Name: {stud\_info\_assign\_subject[2]}")  
 print("\nReturn to home page\n")  
  
  
def stud\_ui(user\_id):  
 stud\_functions = ["View Schedule", "Send Request", "View and Delete Request", "View Payment", "Change Password",  
 "Update Profile", "Log out"]  
 while True:  
 print("------------------------Student Home Page------------------------")  
 for index, function in enumerate(stud\_functions):  
 print(f"Enter {index + 1}: {function}")  
 choice = input("Enter number: ")  
 if choice == "1":  
 stud\_view\_class(user\_id)  
 elif choice == "2":  
 send\_request(user\_id)  
 elif choice == "3":  
 delete\_request(user\_id)  
 elif choice == "4":  
 view\_payment(user\_id)  
 elif choice == "5":  
 change\_password("stud.txt", user\_id)  
 elif choice == "6":  
 update\_profile("stud.txt", user\_id)  
 elif choice == "7":  
 print("Log out success")  
 return "out"  
 else:  
 print("Invalid command, please try again")  
  
  
def stud\_view\_class(user\_id):  
 print("\nHome Page >>> View Class Schedule")  
 print("-" \* 30 + "Class Schedule" + "-" \* 30)  
 stud\_info\_id = find\_one("stud.txt", 0, 1, user\_id)  
 stud\_level = stud\_info\_id[8]  
 stud\_subjects = stud\_info\_id[9:]  
 # Retrieve class schedule that have student's level and subject  
 stud\_class\_list\_level\_subjects = []  
 for stud\_subject in stud\_subjects:  
 stud\_class\_list\_level\_subject = find\_more("class.txt", 4, 6, f"{stud\_level},{stud\_subject}")  
 if not stud\_class\_list\_level\_subject:  
 continue  
 else:  
 for stud\_class\_info\_level\_subject in stud\_class\_list\_level\_subject:  
 stud\_class\_list\_level\_subjects.append(stud\_class\_info\_level\_subject)  
 if not stud\_class\_list\_level\_subjects:  
 print("-"\*30 + "\nNo Class Schedule\n" + "-"\*30 + "\nReturn to home page\n")  
 return  
 # sort all class order by date and time  
 stud\_class\_list\_level\_subjects.sort(key=lambda date: ((date[0], "%d %m %Y"), (date[2], "")))  
 # Display all class  
 for stud\_class\_info\_level\_subjects in stud\_class\_list\_level\_subjects:  
 print(f"Date: {stud\_class\_info\_level\_subjects[0]}\n"  
 f"Venue: {stud\_class\_info\_level\_subjects[1]}\n"  
 f"Time: From {stud\_class\_info\_level\_subjects[2]} "  
 f"to {stud\_class\_info\_level\_subjects[3]}\n"  
 f"Level: {stud\_class\_info\_level\_subjects[4]}\n"  
 f"Subject: {stud\_class\_info\_level\_subjects[5]}\n"  
 f"Charge: {stud\_class\_info\_level\_subjects[6]}\n"  
 f"Lecturer: {stud\_class\_info\_level\_subjects[7]}\n" + "~" \* 100)  
 print("\nReturn to home page\n")  
  
  
def send\_request(user\_id):  
 print("\nHome Page >>> Send Request")  
 print("-" \* 30 + "Send Request" + "-" \* 30)  
 # View levels and subjects  
 view\_levels\_and\_subjects()  
 stud\_info = find\_one("stud.txt", 0, 1, user\_id)  
 stud\_level = stud\_info[8]  
 subjects\_list = retrieve\_level\_or\_subject("subjects", stud\_level)  
 current\_subjects = stud\_info[9:]  
 print(f"(Your current enrolled subject)\nSubject 1: {current\_subjects[0]}\nSubject 2: {current\_subjects[1]}\nSubject 3: {current\_subjects[2]}")  
 if return\_home\_page("send request") is True:  
 return  
 # Get original subject that want to change and identify the validation of the original subject  
 while True:  
 ori\_subject\_num = input("\nEnter the subject number that you want to change (exp: 1/2/3): ")  
 try:  
 ori\_subject\_num = int(ori\_subject\_num)  
 if ori\_subject\_num in range(1, 4):  
 ori\_subject = current\_subjects[ori\_subject\_num-1]  
 break  
 else:  
 print("\nYou must write in 1, 2 or 3 only")  
 if return\_home\_page("re-enter the subject number") is True:  
 return  
 except ValueError:  
 print("\nYou must write in 1, 2 or 3 only")  
 if return\_home\_page("re-enter the subject number") is True:  
 return  
 # Get new subject and identify the validation of the new subject  
 while True:  
 change\_subject = input("Enter subject name that want to change to: ").upper()  
 if change\_subject in subjects\_list and change\_subject not in current\_subjects:  
 break  
 else:  
 print("\nInvalid subject\nPlease enter valid subject")  
 if return\_home\_page("re-enter the subject name") is True:  
 return  
 reason = input("Please provide suitable reason for changing subject: ")  
 request\_list = read\_all("request.txt")  
 if not request\_list:  
 current\_request\_no = "1"  
 else:  
 current\_request\_no = str(int(request\_list[-1][0]) + 1)  
 request\_info = [current\_request\_no, user\_id, ori\_subject, change\_subject, reason, "STILL PENDING"]  
 register("request.txt", request\_info)  
 print("-" \* 30 + "\nRequest has sent to receptionist\n" + "-" \* 30 + "\nReturn to home page\n")  
  
  
def delete\_request(user\_id):  
 print("\nHome Page >>> Delete Request")  
 print("-" \* 30 + "View and Delete Request" + "-" \* 30)  
 stud\_request\_list = find\_more("request.txt", 1, 2, user\_id)  
 if not stud\_request\_list:  
 print("You have no send any request")  
 return  
 else:  
 for request\_info in stud\_request\_list:  
 print(f"Request No.: {request\_info[0]}\nStudent ID: {request\_info[1]}\nFrom Subject: "  
 f"{request\_info[2]}\nChange to Subject: {request\_info[3]}\nReason: "  
 f"{request\_info[4]}\nRequest Status: {request\_info[5]}\n" + "~" \* 30)  
 if return\_home\_page("delete request") is True:  
 return  
 # Get request no. and identify the validation of request no.  
 while True:  
 delete\_line = input("Please enter Request No. for delete: ")  
 if delete\_line.isdigit():  
 request\_info\_request\_no = find\_one("request.txt", 0, 1, delete\_line)  
 if request\_info\_request\_no:  
 if request\_info\_request\_no[-1] in "STILL PENDING":  
 break  
 else:  
 print("You cannot delete this request because it already approved or rejected")  
 if return\_home\_page("re-enter the Request No.") is True:  
 return  
 else:  
 print("Request No. does not exist")  
 if return\_home\_page("re-enter the request no.") is True:  
 return  
 else:  
 print("Invalid Request No.")  
 if return\_home\_page("re-enter the request no.") is True:  
 return  
 delete\_row = int(delete\_line) - 1  
 delete("request.txt", delete\_row)  
 print("-" \* 30 + "\nDelete request success\n" + "-" \* 30 + "\nReturn to home page\n")  
  
  
def view\_payment(user\_id):  
 print("\nHome Page >>> View Payment")  
 print("-"\*30 + "View Payment" + "-"\*30)  
 payment\_list\_id = find\_more("payment.txt", 0, 1, user\_id)  
 payment\_list\_unpaid = find\_more("payment.txt", 10, 11, "UNPAID")  
 payment\_list\_id\_unpaid = []  
 if not payment\_list\_unpaid:  
 print("No payment need to paid")  
 else:  
 for payment\_info\_unpaid in payment\_list\_unpaid:  
 if payment\_info\_unpaid in payment\_list\_id:  
 payment\_list\_id\_unpaid.append(payment\_info\_unpaid)  
 for payment\_info\_id\_unpaid in payment\_list\_id\_unpaid:  
 subject1\_fee = payment\_info\_id\_unpaid[7]  
 subject2\_fee = payment\_info\_id\_unpaid[8]  
 subject3\_fee = payment\_info\_id\_unpaid[9]  
 payable = int(subject1\_fee) + int(subject2\_fee) + int(subject3\_fee)  
 print(f"Date: {payment\_info\_id\_unpaid[1]}\nLevel: {payment\_info\_id\_unpaid[2]}\n"  
 f"Payment Date: {payment\_info\_id\_unpaid[3]}\n"  
 f"Payment Detail: Subject 1 {payment\_info\_id\_unpaid[4]} (RM{subject1\_fee})"  
 f" | Subject 2 {payment\_info\_id\_unpaid[5]} (RM{subject2\_fee})"  
 f" | Subject 3 {payment\_info\_id\_unpaid[6]} (RM{subject3\_fee})\n"  
 f"Payable: RM{payable}\n" + "~" \* 70)  
 print("\nKindly remind: Please pay your education fee before the due date. Thank You\nReturn to home page\n")  
  
  
def change\_password(user\_file, user\_id):  
 print("\nHome Page >>> Change Password")  
 print("-" \* 30 + "Change Password" + "-" \* 30)  
 while True:  
 current\_password = find\_one(user\_file, 0, 1, user\_id)[1]  
 ori\_password = input("original password: ").strip()  
 ori\_password\_crypt = crypt\_password(ori\_password)  
 if ori\_password\_crypt == current\_password:  
 new\_password = input("New password: ").strip()  
 confirm\_password = input("Confirm password: ").strip()  
 if new\_password == confirm\_password:  
 if 8 <= len(new\_password) <= 16:  
 new\_password\_crypt = crypt\_password(new\_password)  
 edit(user\_file, 0, 1, user\_id, 1, new\_password\_crypt)  
 print("-" \* 30 + "\nChange password success\n" + "-" \* 30 + "\nReturn to home page\n")  
 return  
 else:  
 print("Length of your password must minimum 8 and maximum 16")  
 else:  
 print("New password and Confirm password does not match")  
 else:  
 print("Wrong original password")  
 if return\_home\_page("re-enter the password") is True:  
 return  
  
  
def update\_profile(user\_file, user\_id):  
 print("\nHome Page >>> Update Profile")  
 print("-"\*30 + "Update Profile" + "-"\*30)  
 user = find\_one(user\_file, 0, 1, user\_id)  
 print(  
 f"Name: {user[2]}\nIC or Passport: {user[3]}\nAddress: {user[4]}\nEmail: {user[5]}\nContact Number: {user[6]}\n")  
 update\_data = ["Address", "Email", "Contact Number"]  
 if return\_home\_page("update own profile") is True:  
 print("\nReturn to home page\n")  
 return  
 for index, data in enumerate(update\_data):  
 print(f"\nEnter {index + 1}: Update {data}")  
 while True:  
 value = input("\nEnter a number to update the user data: ")  
 try:  
 value = int(value)  
 if 0 < value <= len(update\_data):  
 break  
 else:  
 print("Invalid number")  
 if return\_home\_page("re-enter the number") is True:  
 return  
 except ValueError:  
 print("Invalid user data number")  
 if return\_home\_page("re-enter the number") is True:  
 return  
 update\_item = input("Enter new user data: ")  
 edit(user\_file, 0, 1, user\_id, value + 3, update\_item)  
 print("-"\*30 + "Update profile success" + "-"\*30 + "\nReturn to home page\n")  
  
  
def login():  
 roles\_file = ["admin.txt", "recep.txt", "tutor.txt", "stud.txt"]  
 login\_num = 0  
 while login\_num < 3:  
 user\_id = input("Enter id: ").strip()  
 for current\_file in roles\_file:  
 user\_info = find\_one(current\_file, 0, 1, user\_id)  
 if user\_info:  
 password = user\_info[1]  
 user\_password = input("Enter password: ").strip()  
 user\_password\_crypt = crypt\_password(user\_password)  
 if user\_password\_crypt == password:  
 print("\nLogin success\nWelcome\n")  
 return [current\_file, user\_id]  
 else:  
 print("Invalid Password")  
 if current\_file == "admin.txt":  
 login\_num = -1  
 print("\nInvalid password\nOnly Admin login failed will reset login chance")  
 break  
 else:  
 print("Invalid ID")  
 login\_num += 1  
 print(f"You remain {3 - login\_num} chance !!!\n")  
  
  
def view\_news():  
 with open("news.txt", "r") as news\_file:  
 news\_info = news\_file.readline()  
 if not news\_info:  
 return  
 else:  
 news\_num = 1  
 while news\_info:  
 print(f"News {news\_num}\n")  
 news = news\_info.replace("\\n", "\n")  
 print(news + "~" \* 70)  
 news\_info = news\_file.readline()  
 news\_num += 1  
 return 1  
  
def main():  
 while True:  
 print("-" \* 30 + "Welcome to Brilliant Education Center" + "-" \* 30 + "\n")  
 print("Enter 1: Login account\nEnter 2: View news")  
 while True:  
 choice = input("\nEnter a number: ")  
 if choice == "1":  
 print("\nPlease enter your id and password for login")  
 user\_login = login()  
 if not user\_login:  
 print("You have use all chance and not allow to login")  
 exit()  
 elif user\_login[0] == "admin.txt":  
 admin\_ui(user\_login[1])  
 break  
 elif user\_login[0] == "recep.txt":  
 recep\_ui(user\_login[1])  
 break  
 elif user\_login[0] == "tutor.txt":  
 tutor\_ui(user\_login[1])  
 break  
 else:  
 stud\_ui(user\_login[1])  
 break  
 elif choice == "2":  
 print("\nMain page >>> View news")  
 print("-" \* 30 + "View news" + "-" \* 30)  
 if not view\_news():  
 print("No available of news")  
 print("Return to main page\n")  
 break  
 else:  
 print("Invalid command")  
  
  
main()