****

**Faculty of Engineering and Technology**

**Electrical and Computer Engineering Department**

**Linux Laboratory**

**ENCS3130**

**Python Project**

**My Advisor**

**Prepared by:**

**Mohammad Saeed 1182500**

**Instructor:**

**Dr. Khader Mohammad**

**Teacher assistant:**

**Eng. Kati Sadi**

**Date:**

**17/2/2023**

# **Abstract:**

The aim of this project is to develop a Python script to propose a course schedule for computer engineering university students. The program will analyze the mandatory courses required to obtain a degree, the list of electives, and the courses available in the course browser for the first, second, and summer semesters. It will also consider the courses that the student has already completed, taking into account the maximum number of credits and the number of free days the student wants to register for during each semester. The script will recommend a schedule that aligns with the student's preferences and academic requirements until graduation.

Table Of Contents:

[**Abstract:** 2](#_Toc127568543)

[**1.Theory:** 4](#_Toc127568544)

[1.1. Python 4](#_Toc127568545)

[1.2. Ubuntu 4](#_Toc127568546)

[**2.Procedure and Discussion:** 5](#_Toc127568547)

[2.1. First Student Record txt file 5](#_Toc127568548)

[2.2. Second Student Record txt file 7](#_Toc127568549)

[**3.Conclusion:** 10](#_Toc127568550)

[**4.References:** 11](#_Toc127568551)

[**5.Appendix:** 12](#_Toc127568552)

# **1.Theory:**

## 1.1. Python

**Python** is a [high-level](https://en.wikipedia.org/wiki/High-level_programming_language), [general-purpose programming language](https://en.wikipedia.org/wiki/General-purpose_programming_language). Its design philosophy emphasizes [code readability](https://en.wikipedia.org/wiki/Code_readability) with the use of [significant indentation](https://en.wikipedia.org/wiki/Off-side_rule).

Python is [dynamically typed](https://en.wikipedia.org/wiki/Type_system#DYNAMIC) and [garbage-collected](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)). It supports multiple [programming paradigms](https://en.wikipedia.org/wiki/Programming_paradigm), including [structured](https://en.wikipedia.org/wiki/Structured_programming) (particularly [procedural](https://en.wikipedia.org/wiki/Procedural_programming)), [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) and [functional programming](https://en.wikipedia.org/wiki/Functional_programming). It is often described as a "batteries included" language due to its comprehensive [standard library](https://en.wikipedia.org/wiki/Standard_library).

[Guido van Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum) began working on Python in the late 1980s as a successor to the [ABC programming language](https://en.wikipedia.org/wiki/ABC_(programming_language)) and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000. Python 3.0, released in 2008, was a major revision not completely [backward-compatible](https://en.wikipedia.org/wiki/Backward_compatibility) with earlier versions. Python 2.7.18, released in 2020, was the last release of Python 2 [1].

Python consistently ranks as one of the most popular programming languages.



## 1.2. Ubuntu

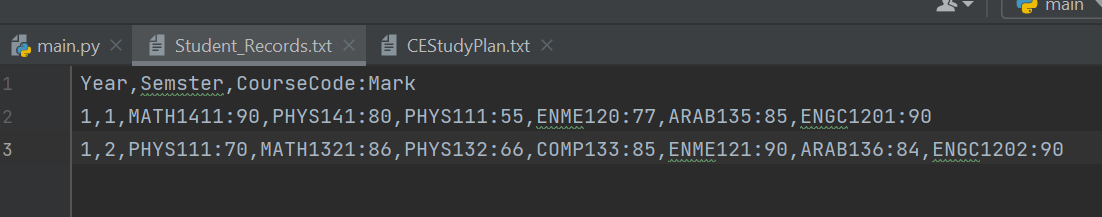
Ubuntu  is a [Linux distribution](https://en.wikipedia.org/wiki/Linux_distribution) based on [Debian](https://en.wikipedia.org/wiki/Debian) and composed mostly of [free and open-source software](https://en.wikipedia.org/wiki/Free_and_open-source_software). Ubuntu is officially released in three editions: [Desktop](https://en.wikipedia.org/wiki/Desktop_computer), [Server](https://en.wikipedia.org/wiki/Server_(computing)), and Core for [Internet of things](https://en.wikipedia.org/wiki/Internet_of_things) devices and [robots](https://en.wikipedia.org/wiki/Robot). All of the editions can run on the computer alone, or in a [virtual machine](https://en.wikipedia.org/wiki/Virtual_machine). Ubuntu is a popular [operating system](https://en.wikipedia.org/wiki/Operating_system) for [cloud computing](https://en.wikipedia.org/wiki/Cloud_computing), with support for [OpenStack](https://en.wikipedia.org/wiki/OpenStack). Ubuntu's default desktop changed back from the in-house [Unity](https://en.wikipedia.org/wiki/Unity_(user_interface)) to [GNOME](https://en.wikipedia.org/wiki/GNOME) after nearly 6.5 years in 2017 upon the release of version [17.10](https://en.wikipedia.org/wiki/Ubuntu_version_history#1710) [2].



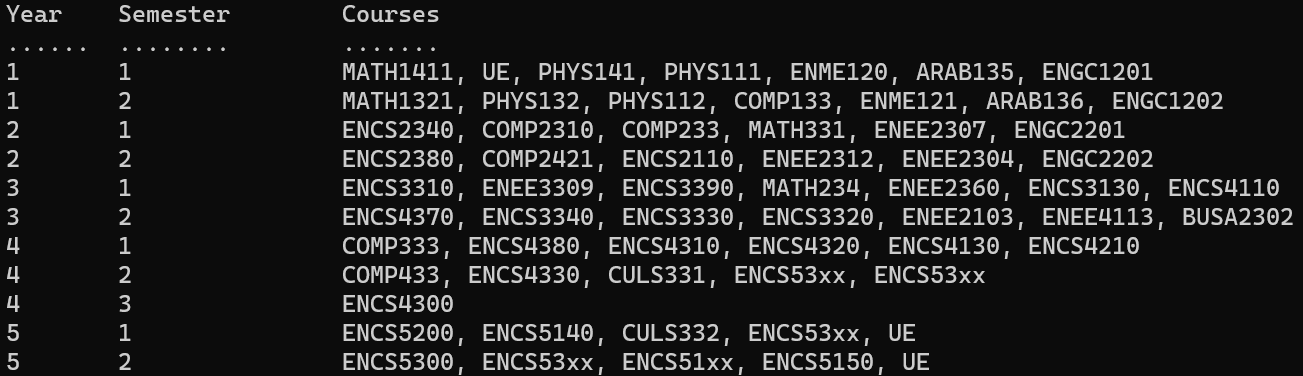
# **2.Procedure and Discussion:**

## 2.1. First Student Record txt file

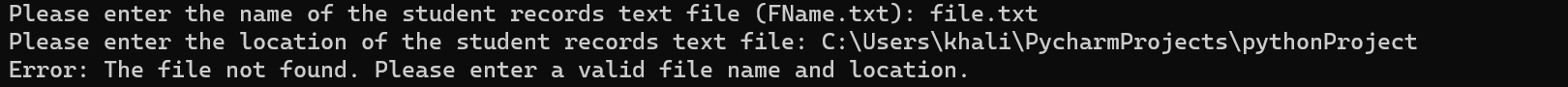
The first student record file shown in the figure below.



First, the program displays the student's Study Plan, providing an overview of the courses the student is required to take for graduation. The plan shown in the figure below.

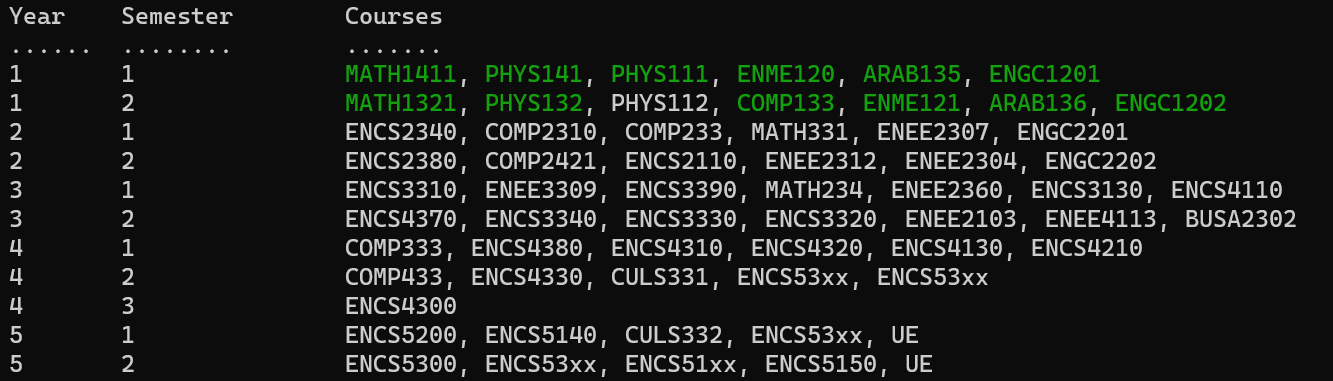


In order to access the Student Records file, the program prompts the user to enter the file's name and location. If the file does not exist, the program will raise an error and prompt the user to re-enter the information.





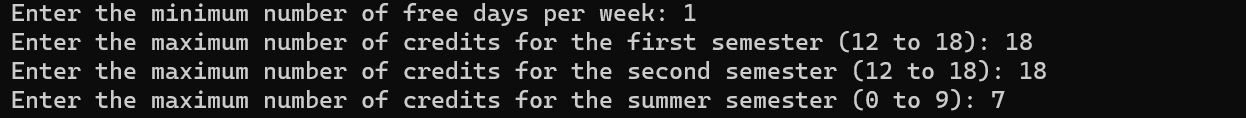
The program presents the student's study plan on the screen, highlighting the courses that the student has already passed with a green color for easy identification.



The project allows for user input of their preferences, which include:

a. Minimum number of free days per week for each semester. (Note that Friday and Sunday are already holidays and are not included as free days. A single free day means the student will attend university for only four days.)

b. Maximum number of credits the student wishes to register for during the first, second, and summer semesters.



The user will be prompted to specify the number of semesters for which the script will generate a course schedule.

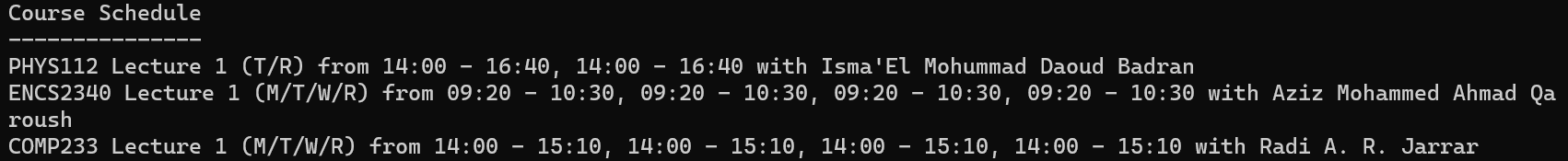


The program will suggest a course schedule for the requested number of semesters or until graduation (if the number of semesters is more than the semesters required to graduate) based on input data such as prerequisites, student studying plan, and day and time of the courses offered, as well as student preferences.

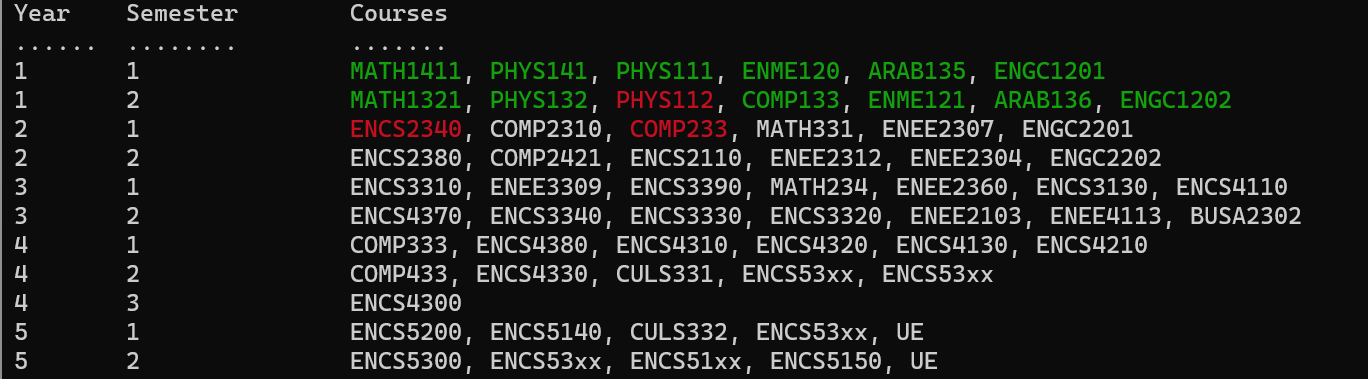
The highest priority for selecting courses must be based on the student's studying plan. The next priority is to fulfill the requirements of courses in the student studying plan in advance. The lowest priority is to achieve the preferences set by the student. If the preferences cannot be achieved, the program should print a message and suggest a course schedule that agrees with the student studying plan but not the preferences.

The program will print on the screen for every semester:

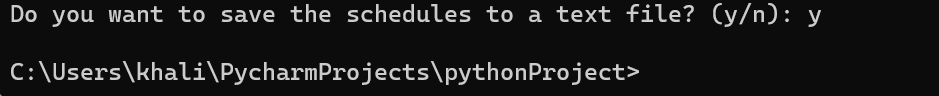
a. the course schedule.

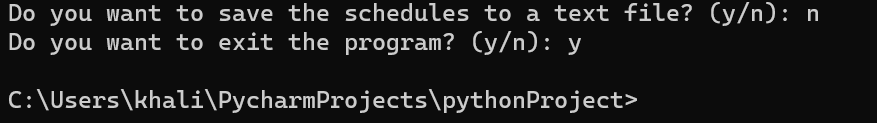


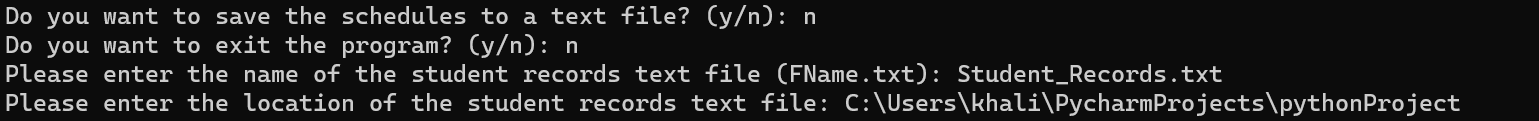
b. the Study Plan such that all courses registered before the current semester are in green color, and the courses scheduled in the current semester are in red color.



In order to save the schedules generated by the program, the script will prompt the user to choose whether to save them to a text file or not. If the user chooses to save, the program will write the schedules to a file named "SuggestedCourses.txt". If the user decides not to save, the program will give the option to either continue or terminate. If the user chooses to continue, the program will return to step 2 in the procedure and prompt the user to input new data. If the user decides to exit, the program will terminate.

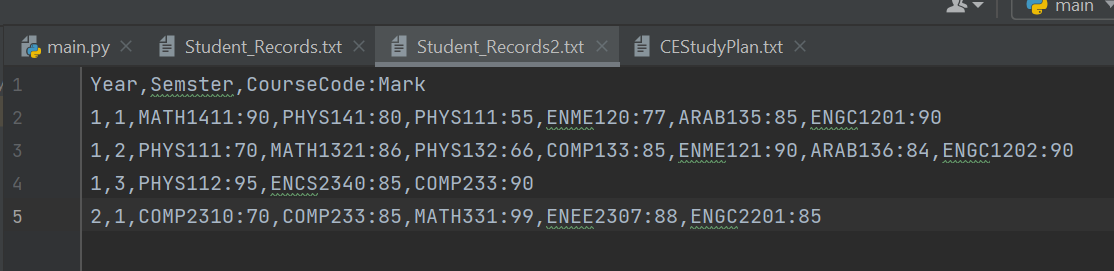




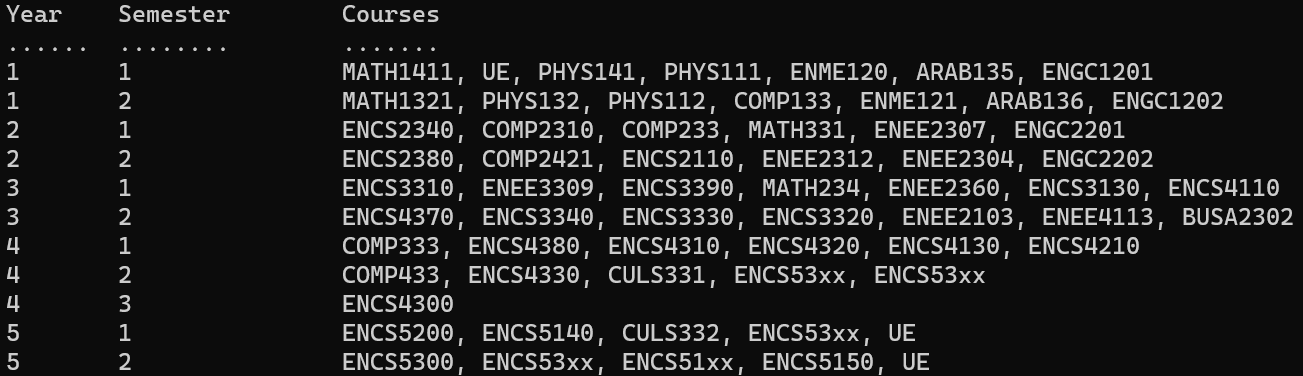


## 2.2. Second Student Record txt file

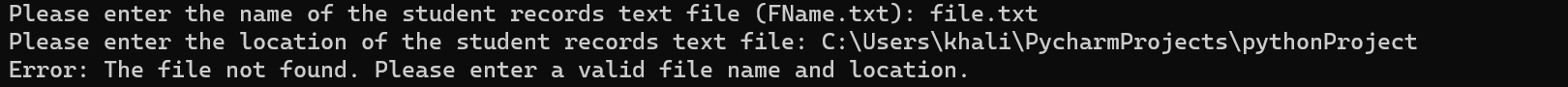
The second student record file shown in the figure below.



First, the program displays the student's Study Plan, providing an overview of the courses the student is required to take for graduation. The plan shown in the figure below.

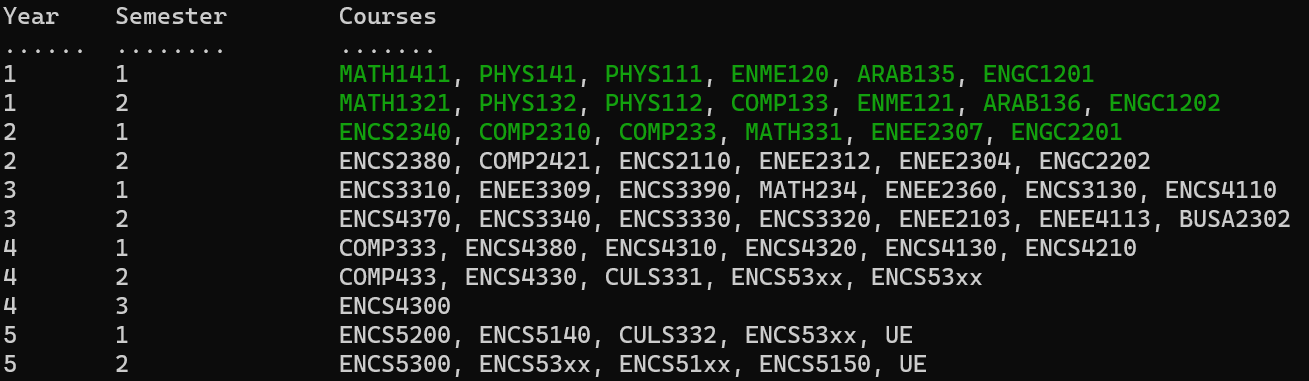


In order to access the Student Records file, the program prompts the user to enter the file's name and location. If the file does not exist, the program will raise an error and prompt the user to re-enter the information.





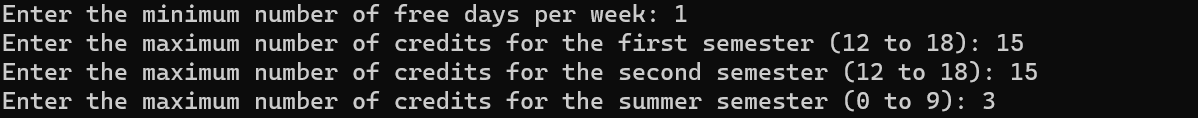
The program presents the student's study plan on the screen, highlighting the courses that the student has already passed with a green color for easy identification.



The project allows for user input of their preferences, which include:

a. Minimum number of free days per week for each semester. (Note that Friday and Sunday are already holidays and are not included as free days. A single free day means the student will attend university for only four days.)

b. Maximum number of credits the student wishes to register for during the first, second, and summer semesters.



The user will be prompted to specify the number of semesters for which the script will generate a course schedule.

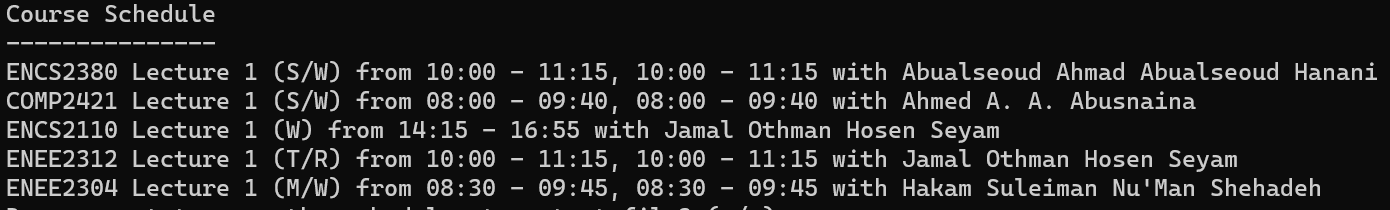


The program will suggest a course schedule for the requested number of semesters or until graduation (if the number of semesters is more than the semesters required to graduate) based on input data such as prerequisites, student studying plan, and day and time of the courses offered, as well as student preferences.

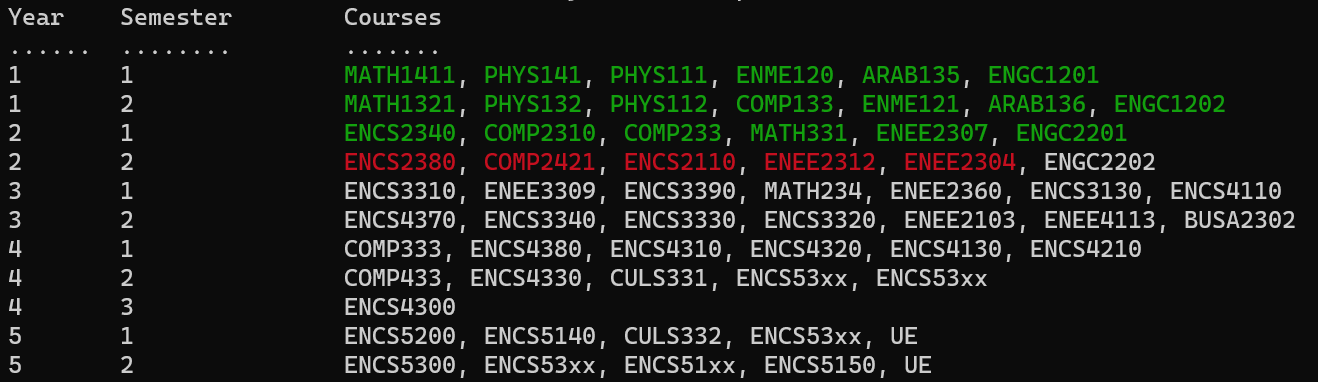
The highest priority for selecting courses must be based on the student's studying plan. The next priority is to fulfill the requirements of courses in the student studying plan in advance. The lowest priority is to achieve the preferences set by the student. If the preferences cannot be achieved, the program should print a message and suggest a course schedule that agrees with the student studying plan but not the preferences.

The program will print on the screen for every semester:

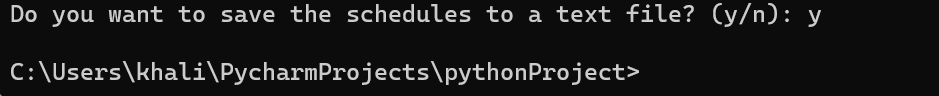
a. the course schedule.

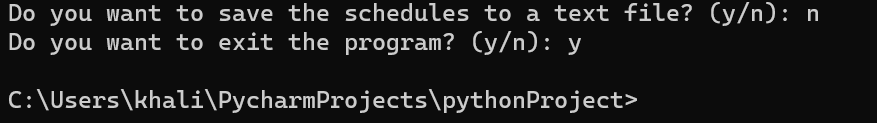


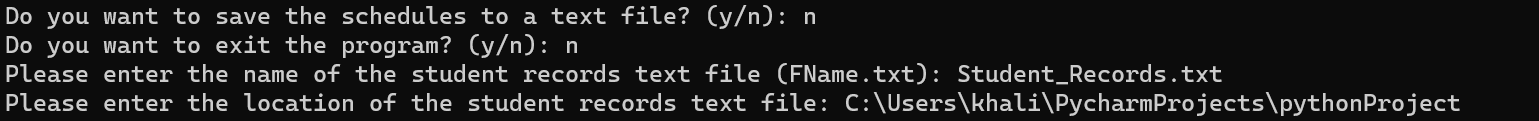
b. the Study Plan such that all courses registered before the current semester are in green color, and the courses scheduled in the current semester are in red color.



In order to save the schedules generated by the program, the script will prompt the user to choose whether to save them to a text file or not. If the user chooses to save, the program will write the schedules to a file named "SuggestedCourses.txt". If the user decides not to save, the program will give the option to either continue or terminate. If the user chooses to continue, the program will return to step 2 in the procedure and prompt the user to input new data. If the user decides to exit, the program will terminate.







# **3.Conclusion:**

In conclusion, the python script developed in this project provides a useful tool for computer engineering students to plan their course schedules until graduation. By parsing the study plan, list of elective courses, available courses, and student records, the script generates a personalized course schedule based on the student's requirements and preferences. The flexibility of the script allows the student to set the maximum number of credits and the number of free days per week for each semester. The script can be further extended to support other majors and courses to provide assistance for more students in their academic journey.

# **4.References:**

[1]: <https://en.wikipedia.org/wiki/Python_(programming_language)> , [Accessed on 17/2/2023 13:21]

[2]: <https://en.wikipedia.org/wiki/Ubuntu> , [Accessed on 17/2/2023 13:44]

# **5.Appendix:**

import csv  
import os  
import termcolor  
import json  
  
# this function to calculate the cridits of the course  
def course\_credits(course\_code):  
 if course\_code == "UE":  
 return 3  
 return int(course\_code[5])  
  
# this function to choose the courses for the next semester  
def choose\_courses(study\_plan, max\_credits,CourseBrowser):  
 not\_passed\_courses = []  
  
 # Loop over all courses in the study plan and check if they're not passed  
 for year\_data in study\_plan.values():  
 for semester\_data in year\_data.values():  
 for course in semester\_data:  
 if not course['is\_passed']:  
 # If the course is not passed, add it to the list of not passed courses  
 not\_passed\_courses.append(course)  
  
 # Return the first not passed courses that satisfies the max credits requirement  
 selected\_courses = []  
 total\_credits = 0  
 for course in not\_passed\_courses:  
 credits = int(course\_credits(course['course\_code']))  
 if total\_credits + credits <= max\_credits:  
 # Check in the course exist in the course browser  
 for key in CourseBrowser:  
 if course['course\_code'] in key:  
 selected\_courses.append(course['course\_code'])  
 total\_credits += credits  
 break  
 else:  
 break  
 return selected\_courses  
  
# print the study plan after choosing the courses  
def print\_study\_plan(study\_plan, selected\_courses):  
 print("Year\tSemester\tCourses")  
 print("......\t........\t.......")  
 for year, semesters in study\_plan.items():  
 for semester, courses in semesters.items():  
 courses\_str = ', '.join([  
 termcolor.colored(c['course\_code'], 'green') if c['is\_passed'] and c['course\_code'] not in selected\_courses  
 else termcolor.colored(c['course\_code'], 'red') if c['course\_code'] in selected\_courses  
 else c['course\_code'] for c in courses  
 ])  
 print(f"{year}\t{semester}\t\t{courses\_str}")  
  
# This function to print the schedule of the courses  
def print\_schedule(selected\_courses,CourseBrowser):  
 with open(CourseBrowser) as f:  
 course\_schedule = json.load(f)  
  
 #print("The preferences cannot be achieved, I suggest a course schedule that agrees with the student studying plan but not the preferences.")  
 print('Course Schedule\n---------------')  
 selected\_sections = []  
 for course in selected\_courses:  
 for section\_type in ['Lecture', 'Lab']:  
 for section\_num in range(1, 4):  
 section\_key = f"{course}-{section\_type}-{section\_num}"  
 if section\_key in course\_schedule and section\_key not in selected\_sections:  
 course\_info = course\_schedule[section\_key]  
 instructor = course\_info.get('Instructor')  
 days = [day for day in course\_info.keys() if day != 'Instructor']  
 times = [course\_info[day] for day in days]  
 days\_str = '/'.join(days)  
 times\_str = ', '.join(times)  
 print(f"{course} {section\_type} {section\_num} ({days\_str}) from {times\_str} with {instructor}")  
 selected\_sections.append(section\_key)  
 break  
 else:  
 continue  
 break  
 else:  
 print(f"{course} is not offered this semester.")  
  
# Print schedule in the SuggestedCourses.txt file  
def print\_schedule\_on\_file(selected\_courses, CourseBrowser):  
 with open(CourseBrowser) as f:  
 course\_schedule = json.load(f)  
  
 with open('SuggestedCourses.txt', 'w') as f:  
 f.write('Course Schedule\n---------------\n')  
 selected\_sections = []  
 for course in selected\_courses:  
 for section\_type in ['Lecture', 'Lab']:  
 for section\_num in range(1, 4):  
 section\_key = f"{course}-{section\_type}-{section\_num}"  
 if section\_key in course\_schedule and section\_key not in selected\_sections:  
 course\_info = course\_schedule[section\_key]  
 instructor = course\_info.get('Instructor')  
 days = [day for day in course\_info.keys() if day != 'Instructor']  
 times = [course\_info[day] for day in days]  
 days\_str = '/'.join(days)  
 times\_str = ', '.join(times)  
 f.write(f"{course} {section\_type} {section\_num} ({days\_str}) from {times\_str} with {instructor}\n")  
 selected\_sections.append(section\_key)  
 break  
 else:  
 continue  
 break  
 else:  
 f.write(f"{course} is not offered this semester.\n")  
  
study\_plan = {}  
# Read the data from the file and store it in dictionary  
with open('CEStudyPlan.txt') as file:  
 reader = csv.DictReader(file)  
 for row in reader:  
 year = row['Year']  
 semester = row['Semster']  
 course\_code = row['CourseCode']  
 prerequisites = row['Prerequisists']  
 is\_passed = False  
  
 if year not in study\_plan:  
 study\_plan[year] = {}  
 if semester not in study\_plan[year]:  
 study\_plan[year][semester] = []  
 course = {'course\_code': course\_code, 'is\_passed': is\_passed, 'prerequisites': prerequisites}  
 study\_plan[year][semester].append(course)  
  
# Print the study plan  
print('Year\tSemester\tCourses')  
print('......\t........\t.......')  
for year, semesters in study\_plan.items():  
 for semester, courses in semesters.items():  
 courses\_str = ', '.join([c['course\_code'] for c in courses])  
 print(f'{year}\t{semester}\t\t{courses\_str}')  
  
  
while True:  
  
 last\_semester = 0 # To store the number of last semester which exist in the student record  
  
 # Reach to the Student\_Records.txt file and read the data from it  
 while True:  
 file\_name = input("Please enter the name of the student records text file (FName.txt): ")  
 file\_path = input("Please enter the location of the student records text file: ")  
 full\_path = os.path.join(file\_path, file\_name)  
 try:  
 # Read the data from the Student\_Records.txt file  
 with open(full\_path, "r") as file:  
 reader = csv.reader(file)  
 next(reader) # skip the header row  
 for row in reader:  
 year, semester, course\_marks = row[0], row[1], row[2:]  
 last\_semester = int(semester) # To store the last semester which exist the student record  
 for course\_mark in course\_marks:  
 course\_code, mark = course\_mark.split(":")  
 mark = int(mark)  
 if mark >= 60:  
 for year\_data in study\_plan.values():  
 for semester\_data in year\_data.values():  
 for course in semester\_data:  
 if course['course\_code'] == course\_code:  
 course['is\_passed'] = True  
 break  
 except FileNotFoundError:  
 print("Error: The file not found. Please enter a valid file name and location.")  
  
 # Print the Study Plan on the screen and identify the passed courses in green color  
 print("Year\tSemester\tCourses")  
 print("......\t........\t.......")  
 for year, semesters in study\_plan.items():  
 for semester, courses in semesters.items():  
 courses\_str = ', '.join(  
 [termcolor.colored(c['course\_code'], 'green') if c['is\_passed'] else c['course\_code'] for c in courses])  
 print(f"{year}\t{semester}\t\t{courses\_str}")  
  
 # Ask the user to enter the minimum number of free days per week  
 min\_free\_days = int(input("Enter the minimum number of free days per week: "))  
  
 # Ask the user to enter the maximum number of credits per semester  
 max\_credits\_first\_semester = int(input("Enter the maximum number of credits for the first semester (12 to 18): "))  
 while max\_credits\_first\_semester < 12 or max\_credits\_first\_semester > 18:  
 print("Invalid input. Please enter a number between 12 and 18.")  
 max\_credits\_first\_semester = int(  
 input("Enter the maximum number of credits for the first semester (12 to 18): "))  
  
 max\_credits\_second\_semester = int(input("Enter the maximum number of credits for the second semester (12 to 18): "))  
 while max\_credits\_second\_semester < 12 or max\_credits\_second\_semester > 18:  
 print("Invalid input. Please enter a number between 12 and 18.")  
 max\_credits\_second\_semester = int(  
 input("Enter the maximum number of credits for the second semester (12 to 18): "))  
  
 max\_credits\_summer\_semester = int(input("Enter the maximum number of credits for the summer semester (0 to 9): "))  
 while max\_credits\_summer\_semester < 0 or max\_credits\_summer\_semester > 9:  
 print("Invalid input. Please enter a number between 0 and 9.")  
 max\_credits\_summer\_semester = int(  
 input("Enter the maximum number of credits for the summer semester (0 to 9): "))  
  
 # Ask the user about the number of semesters  
 while True:  
 num\_semesters = int(  
 input("Enter the number of semesters for which you want to plan the schedule (minimum 1 semester): "))  
 if num\_semesters > 0:  
 break  
 else:  
 print("Invalid input. The number of semesters should be greater than 0. Please try again.")  
  
 #############################################################  
 # Read the data of course browser for the first semester  
 with open('CourseBrowser\_1.json', 'r') as f:  
 CourseBrowser\_1 = json.load(f)  
  
 # Read the data of course browser for the second semester  
 with open('CourseBrowser\_2.json', 'r') as f:  
 CourseBrowser\_2 = json.load(f)  
  
 # Read the data of course browser for the summer semester  
 with open('CourseBrowser\_3.json', 'r') as f:  
 CourseBrowser\_3 = json.load(f)  
 #############################################################  
 current\_semester = last\_semester + 1 # To store the number of the current semester  
 selected\_courses = []  
  
 if current\_semester == 1:  
 selected\_courses = choose\_courses(study\_plan, max\_credits\_first\_semester,CourseBrowser\_1)  
 elif current\_semester == 2:  
 selected\_courses = choose\_courses(study\_plan, max\_credits\_second\_semester,CourseBrowser\_2)  
 elif current\_semester == 3:  
 selected\_courses = choose\_courses(study\_plan, max\_credits\_summer\_semester,CourseBrowser\_3)  
 #########################################################################  
 # Print the study plan with the next semester courses in the red color  
 print\_study\_plan(study\_plan, selected\_courses)  
 ########################################  
 # Print the Schedule in terms of the semester  
 if current\_semester == 1:  
 print\_schedule(selected\_courses,"CourseBrowser\_1.json")  
 elif current\_semester == 2:  
 print\_schedule(selected\_courses,"CourseBrowser\_2.json")  
 elif current\_semester == 3:  
 print\_schedule(selected\_courses,"CourseBrowser\_3.json")  
 #######################################  
  
 flag = False # To do (continue) to the main loop  
 # final part,print the data in the file and exit  
 while True:  
 save\_to\_file = input("Do you want to save the schedules to a text file? (y/n): ")  
 if save\_to\_file.lower() == 'y':  
 # Print the Schedule on the file in terms of the semester  
 if current\_semester == 1:  
 print\_schedule\_on\_file(selected\_courses, "CourseBrowser\_1.json")  
 elif current\_semester == 2:  
 print\_schedule\_on\_file(selected\_courses, "CourseBrowser\_2.json")  
 elif current\_semester == 3:  
 print\_schedule\_on\_file(selected\_courses, "CourseBrowser\_3.json")  
 break  
 elif save\_to\_file.lower() == 'n':  
 exit\_program = input("Do you want to exit the program? (y/n): ")  
 if exit\_program.lower() == 'y':  
 break  
 elif exit\_program.lower() == 'n':  
 flag = True  
 break  
 else:  
 print("Invalid input.")  
 break  
 else:  
 print("Invalid input. Please enter 'y' or 'n'.")  
  
 if flag:  
 continue  
 break