빅데이터 AI 아카데미 12 기

성적 관리 프로그램

파이썬 1 주차 과제

이름:

김준오

이메일 : juno1028@naver.com

명예서약(Honor code)

"나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다."

Problem 1: 성적 관리 프로그램

1. 문제의 개요

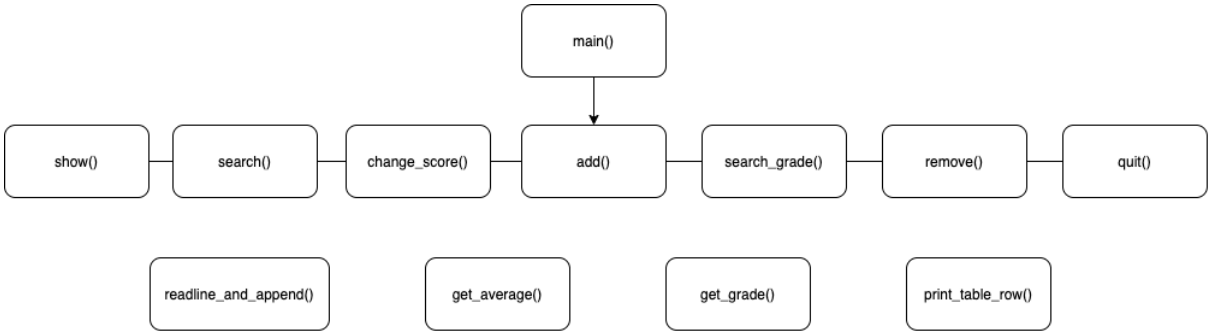
본 프로그램을 간략히 설명하면 다음과 같다.

(문제)

파일로부터 데이터를 읽어서 성적 목록을 만들어 관리하는 성적 관리 프로그램을 작성한다. 7 개의 명령어(show, search, changescore, searchgrade, add, remove, quit)를 입력 받아 각 기능을 수행 하게 된다.

- **Show** (전체 학생 정보 출력) : show 입력 시, 저장되어 있는 전체 목록을 아래와 같이 평균 점수를 기준으로 내림차순으로 출력한다. 평균 점수는 소수점 이하 첫째 자리까지만 표시한다.
- **Search** (특정 학생 검색) : search 입력 시, 아래와 같이 검색하고자 하는 학생의 학번을 요구해 입력 받아 학번, 이름, 중간고사 점수, 기말고사 점수, 평균, 학점을 출력한다.
- **Changescore** (점수 수정) : 목록에 저장된 학생 중 1 명의 중간고사(mid) 혹은 기말고사(final)의 점수를 수정한다.
- **Add** (학생 추가) : 학생의 학번, 이름, 중간고사 점수, 기말고사 점수를 차례로 요구해 입력 받는다.
- **Searchgrade** (Grade 검색) : 특정 grade 를 입력 받아 그 grade 에 해당하는 학생을 모두 출력한다.
- **Remove** (특정 학생 삭제) : 삭제하고자 하는 학생의 학번을 입력 받은 후, 학생이 목록에 있는 경우 삭제한다.
- **Quit** (종료) : 프로그램을 종료한다. 현재까지 편집한 내용의 저장 여부를 묻고, 저장을 선택할 경우 파일명을 입력 받아서 저장하도록 한다.

이 때 사용되는 구상 가능한 구조 차트(structure chart)는 아래와 같이 표현될 수 있다.



- 입력부: 학생들의 학번, 이름, 중간고사 점수, 기말고사 점수가 들어있는 텍스트 파일을 입력으로 한다. 파일이 지정되어 있지 않다면, 기본적으로 동일경로 내에 있는 students.txt 파일을 읽어오도록 한다.
- 처리부: show(), search(), change_score(), add(), search_grade(), remove(), quit()의 총 7 가지의 처리 함수를 가지고 있다. 모두 해당 함수의 이름을 명령어로 입력받아 실행된다.
- 출력부: 처리가 끝난 데이터를 사용자가 원할 시, 파일 형식으로 출력한다. 사용자가 quit 명령어를 입력하면, 파일 저장 여부를 물어보고 실행하도록 한다.
- Readline_and_append(), get_average(), get_grade(), print_table_row()는 처리부 함수 내부에 반복적으로 들어가는 코드를 따로 함수화하여 놓아서 코드의 가독성을 높이고 반복을 줄이는데에 활용한다.

2. 알고리즘

본 프로그램 작성을 위한 알고리즘을 Pseudo 코드 형태로 나타내면 다음과 같다.

Pseudo-algorithm for main()

// 프로그램에 필요한 변수들은 미리 선언해놓은 것으로 가정한다.

```
while(True):
    # 파일 읽어오기 및 데이터 저장하기
    if sys.argv's length == 1:  # 사용자가 뒤에 아무것도 입력하지
않았을 때,
        file_name = ""
    else:
```

```
        set file_name as sys.argv[1]  # 사용자가 뒤에 파일명을
입력했을 때,
    # stu_list 를 선언해준다. 나중에 stu_list = [["20180001","Hong
Gildong", 84, 73], ["20180001","Hong Gildong", 84, 73], ...]
    stu_list = []

    # input 받은 파일명에 따라 실행되는거 분류

    if file_name == "":
        open students.txt as fr:
            readline_and_append(fr)
            break

    elif file_name exists in path:
        open students.txt as fr:
            readline_and_append(fr)
            break
    else:
        print 'File not Found'
        # 종료 시키기
        Quit the program

# stu_list 에 Average, Grade 추가하기
# [["20180001","Hong Gildong", 84, 73] , [~~], ...] -->
[["20180001","Hong Gildong", 84, 73, 78.5, "C"], [~~],...]
for student in stu_list:
    set student_average_score = get_average(student_mid_score,
student_fina_score)
    student_grade = get_grade((mid_score + final_score) / 2)
```

```
    append student_grade to student
    append student to stu_list

# stu_list 평균 점수를 기준으로 내림차순 정렬
Sort stu_list by average_score(reverse=True)

# 프로그램 실행시키면 출력되는 화면

print_table_row # 열과 표를 그려주는 함수
for student in stu_list:
    print student's info

# 명령어 기다리기

while(True):
    set input_command by input()
    make this word upper
    if input_command == "SHOW":
        show()
    elif input_command == "SEARCH":
        search()
    elif input_command == "CHANGESCORE":
        change_score()
    elif input_command == "ADD":
        add()
    elif input_command == "SEARCHGRADE":
        search_grade()
    elif input_command == "REMOVE":
        remove()
```

```
    elif input_command == "QUIT":
        quit()
        get out of while # while 문을 break 하여 빠져나온다.
    else:
        continue # 계속해서 돌아간다.
```

Pseudo-algorithm show()

// 프로그램에 필요한 변수들은 미리 선언해놓은 것으로 가정한다.

```
def show():
    print_table_row()
    for student in stu_list:
        print student # student 안에는 학번, 이름, 중간고사성적, 기말고사 성적, 평균, 학점이 들어가있다.
    return
```

Pseudo-algorithm search()

// 프로그램에 필요한 변수들은 미리 선언해놓은 것으로 가정한다.

```
def search():
    set student_id user_input
    for student in stu_list:
        if student's id == user_input_student_id
            print_table_row()
            print student's information
```

```
            return  # 찾으면 함수를 종료시켜 버린다.
    print "NO SUCH PERSON."  # for 문을 다 돌때까지 찾지 못하면
"NO SUCH PERSON."을 띄운다.
    return
```

Pseudo-algorithm change_score()

// 프로그램에 필요한 변수들은 미리 선언해놓은 것으로 가정한다.

```
def change_score():
    set student_id user_input
    for student in stu_list:
        if there is student_id match to student_list
            ask mid_or_final
            if answer is mid
                ask new_score
                if valid score
                    # 바뀌기 이전 score 출력
                    print_table_row()
                    print student information

                    student's mid_score = new_score  # 중간고사 점수
수정
                    student's average score = get_average(
                        mid score, final score)  # 평균점수 다시 계산
                    recalculate the grade by new scores  # Grade 다시
계산
```

```
                # 바뀐 후 score 출력
                print "Score changed."
                print new_student_info

                # 전체 리스트 다시 정렬
                rearrage the list
                return
            else:
                return

        elif answer is final:  # 기말고사일 때,
            ask new score
            if valid score:
                student's final score = new_score  # 기말고사 점수
수정
                student's average score = get_average(
                    student_mid, student_final)  # 평균점수 다시 계산
                recalculate the grade  # Grade 다시 계산
                # 전체 리스트 다시 정렬
                rearrange the entire list
                return
            else:
                return
        else:
            return

    print "NO SUCH PERSON."   # for 문을 다 돌때까지 찾지 못하면
"NO SUCH PERSON."을 띄운다.
    return
```

Pseudo-algorithm add()

// 프로그램에 필요한 변수들은 미리 선언해놓은 것으로 가정한다.

```
def add():
    existing_id_list
    for student in stu_list:
        append id to existing_id_list  # 이미 존재하는 학번 리스트를 만든다.
    set input_id by user_input
    # 이미 학번 존재하면 함수를 return 해버린다.
    if input_id in existing_id_list:
        print "ALREADY EXISTS"
        return
    # 학번이 중복되지 않는 것이 확인되었으므로, 나머지 정보도 입력받는다.
    set input_name by user_input
    set input_midterm_score by int(user_input)
    input_final_score = int(user_input)
    # 평균과 학점을 계산한다.
    calculate averge_score = get_average(input_midterm_score, input_final_score)
    grade = get_grade(averge_score)
    # 새로운 학생의 정보 리스트를 만든다.
```

```
    set new_student_info_list with
        [input_id, input_name, input_midterm_score, input_final_score,
averge_score, grade]
    # stu_list 에 추가한다.
    stu_list.append(new_student_info_list)
    # 전체 학생 리스트를 평균 순으로 다시 정렬한다.
    rearrange_entire_stu_list by average_score
    print "Student added."
    return
```

Pseudo-algorithm search_grade()

// 프로그램에 필요한 변수들은 미리 선언해놓은 것으로 가정한다.

```
def search_grade():
    existing_grade_list = []
    for student in stu_list:
        if student's id does not exist in in existing_grade_list
            append student id to existing_grade_list # 이미 존재하는
학번 리스트를 중복없이 만든다.
    # 찾고있는 학점을 입력받는다.
    set grade_to_search by user_input
    # "A", "B", "C", "D", "F" 중 하나가 아니라면, return 한다.
    if grade_to_search not in ["A", "B", "C", "D", "F"]:
        return
    # 학생들의 학점 중에 없다면, return 한다.
    elif grade_to_search does not exist in existing_grade_list:
        print "NO RESULTS."
```

```
        return
    else:
        searching_grade_student_list = []
        for student in stu_list:
            if student's grade same with grade_to_search:
                append student to searching_grade_student_list
        for student in searching_grade_student_list:
            print student info
        return
    return
```

Pseudo-algorithm remove()

// 프로그램에 필요한 변수들은 미리 선언해놓은 것으로 가정한다.

```
def remove():
    # 목록에 아무도 없을 경우, return 한다.
    if list is empty:
        print("List is empty.")
        return

    # 학번 리스트를 만든다.
    existing_id_list = []
    for student in stu_list:
        append student id to existing_id_list

    # 학번을 입력받는다.
```

```
    set input_id by user_input_id
    if input_id not in existing_id_list:
        print "NO SUCH PERSON."
        return
    else:
        for student in stu_list:
            if student id same to input_id:
                stu_list.remove(student)
                break
        print("Student removed.")
        return
```

Pseudo-algorithm quit()

// 프로그램에 필요한 변수들은 미리 선언해놓은 것으로 가정한다.

```
def quit():
    ask yes_or_no and set save_data_yes_or_no by user_input
    if save_data_yes_or_no == "yes":
        ask file_name_to_write and set variable by user_input
        open file to write by file name : file_name_to_write
        for student in stu_list:
            print student info
            write data to writing file
        close file
    return
```

Pseudo-algorithm readline_and_append(readed_file)

// 프로그램에 필요한 변수들은 미리 선언해놓은 것으로 가정한다.

```
def readline_and_append(readed_file):
    for line in readed_file:
        set first name from line
        set full name from line
        # ["20180001","Hong Gildong", 84, 73]
        stu_number_name_mid_final_list = [id, full name, mid_score, final_score]
        # gildong = ["20180001","Hong Gildong", 84, 73] 형태로도 저장됨
        first_name = stu_number_name_mid_final_list  # 변수값을 리스트의 변수명으로 사용할 예정 -> 찾아봐야할 듯
        # 중첩 리스트 형식으로도 저장
        append stu_number_name_mid_final_list to stu_list
    return
```

Pseudo-algorithm get_average(mid, final)

// 프로그램에 필요한 변수들은 미리 선언해놓은 것으로 가정한다.

```
def get_average(mid, final):
    return (mid + final)/2
```

Pseudo-algorithm get_grade(score)

// 프로그램에 필요한 변수들은 미리 선언해놓은 것으로 가정한다.

```
def get_grade(score):
    if score >= 90:
        return 'A'
    elif score >= 80 and score < 90:
        return 'B'
    elif score >= 70 and score < 80:
        return 'C'
    elif score >= 60 and score < 70:
        return 'D'
    else:
        return "F"
```

Pseudo-algorithm print_table_row()

// 프로그램에 필요한 변수들은 미리 선언해놓은 것으로 가정한다.

```
def print_table_row():
    print("Student" + "₩t₩t" + "Name" + "₩t₩t" + "Midterm" + "₩t"
+
          "Final" + "₩t" + "Average" + "₩t" + "Grade")
```

```
    print("-" * 70)
    return
```

---

3. 프로그램 구조 및 설명

a) main()

```
##################### 실행되는 부분(메인함수) ####################


while(True):
    # 파일 읽어오기 및 데이터 저장하기
    if len(sys.argv) == 1:  # 사용자가 뒤에 아무것도 입력하지 않았을 때,
        file_name = ""
    else:
        file_name = sys.argv[1]  # 사용자가 뒤에 파일명을 입력했을 때,
    # stu_list를 선언해준다. 나중에 stu_list = [["20180001","Hong Gildong", 84, 73], ["20180001","Hong Gildong",
    stu_list = []

    # input 받은 파일명에 따라 실행되는거 분류

    if file_name == "":
        with open("students.txt", "r") as fr:
            readline_and_append(fr)
            break

    elif os.path.exists(file_name):
        with open(file_name, "r") as fr:
            readline_and_append(fr)
            break
    else:
        print('File not Found')
        # 종료 시키기
        sys.exit()

# stu_list에 Average, Grade 추가하기
# [["20180001","Hong Gildong", 84, 73] , [~~], ...] --> [["20180001","Hong Gildong", 84, 73, 78.5, "C"], [~~],...
for student in stu_list:
    student_average_score = get_average(student[2], student[3])
    student_grade = get_grade(student_average_score)
    student.append(student_average_score)
    student.append(student_grade)

# stu_list 평균 점수를 기준으로 내림차순 정렬
stu_list.sort(key=lambda e: e[4], reverse=True)

# 프로그램 실행시키면 출력되는 화면

print_table_row()
for student in stu_list:
    print(student[0] + "\t" + student[1] + "\t" + str(student[2]) + "\t" +
          str(student[3]) + "\t" + str(student[4]) + "\t" + student[5])
```

```
# 명령어 기다리기

while(True):
    upper_input_command = input("# ").upper()
    if upper_input_command == "SHOW":
        show()
    elif upper_input_command == "SEARCH":
        search()
    elif upper_input_command == "CHANGESCORE":
        change_score()
    elif upper_input_command == "ADD":
        add()
    elif upper_input_command == "SEARCHGRADE":
        search_grade()
    elif upper_input_command == "REMOVE":
        remove()
    elif upper_input_command == "QUIT":
        quit()
        break
    else:
        continue
```

- 프로그램을 실행하면, 실행되는 부분으로, show()부터 quit() 함수까지 7 가지 함수를 포함한다. 처음 파일을 읽어올 때, sys.argv 의 길이가 1, 즉 사용자가 .py 파일 뒤에 아무것도 입력하지 않았을 때에는 file_name 을 빈 문자열로 선언한다. Sys.argv 의 길이가 1 이 아니라면, 사용자가 입력한 문자열을 file_name 으로 선언한다.

- file_name 이 비어있는 문자열이면, students.txt 를 열고, file_name 이 빈 문자열이 아닐 경우에는 os.path.exists()를 통해 경로에 파일이 존재하는지 판단하고 있으면 파일을 읽어온 다음, readline_and_append() 함수를 통해, stu_list 에 student 정보를 담는다.

- 사용자가 입력한 파일명을 가진 파일이 없다면, "File not Found"를 출력하고 프로그램을 종료한다.

- stu_list 를 for 문을 돌려서 student 를 하나씩 가져와서, student_average_score 를 get_average() 함수를 활용해 구하고, student_grade 를 get_grade 함수를 활용해 구한다. 그 다음, student 에 student_average_score 와 student_grade 를 append 해준다.

- Stu_list 를 lambda 함수를 활용해서 student_average_score 를 key 로 써서 내림차순(reverse = True)으로 정렬한다.
- Print_table_row()를 통해, 표의 윗부분을 출력하고, for 문을 통해 stu_list 의 student 의 정보를 한 줄씩 차례대로 출력한다.
- While(True)를 통해서 실행할 명령어를 받는데, upper_input_command 로 사용자의 input 을 대문자화하여 받는다.
- 각 upper_input_command 에 매치되는 함수가 실행될 것이고, quit 이 입력되면, break 를 통해 반복문을 빠져나오게 된다.

b) show()

```
###################### show 함수 ######################


def show():
    print_table_row()
    for student in stu_list:
        print(student[0] + "\t" + student[1] + "\t" + str(student[2]) + "\t" +
              str(student[3]) + "\t" + str(student[4]) + "\t" + student[5])
    return
```

- stu_list 에 저장되어있는 반복문을 통해 student 요소를 한 줄씩 출력해준다. 0 번 인덱스부터 출력하여, 평균이 높은 학생들부터 출력된다.

- 평균이 높은 학생부터 출력할 수 있는 이유는 stu_list 가 이미 평균을 기준으로 내림차순 정렬이 되어있는 상태이기 때문이다.

- "₩t"와 "₩n"을 적절히 활용하여 적당한 간격으로 출력한다.

c) search()

```
###################### search 함수 ######################


def search():
    student_id = input('Student ID: ')
    for student in stu_list:
        if student[0] == student_id:
            print_table_row()
            print(student[0] + "\t" + student[1] + "\t" + str(student[2]) + "\t" +
                  str(student[3]) + "\t" + str(student[4]) + "\t" + student[5])
            return  # 찾으면 함수를 종료시켜 버린다.
    print("NO SUCH PERSON.")  # for 문을 다 돌때까지 찾지 못하면 "NO SUCH PERSON."을 띄운다.
    return
```

- student_id 를 사용자에게 입력받고, stu_list 를 for 문으로 돌면서 student_id 와 일치하는 student 를 찾는다. Student_id 는 고유한 값이므로, for 문을 돌다가 일치하는 것을 찾게되면 표와 학생 정보를 출력하고 return 으로 함수 자체를 종료시킨다.
- 일치하는 student_id 를 for 문을 다 돌때까지 찾지 못하면, "NO SUCH PERSON"을 출력하면서 함수가 끝난다.

d) change_score()

```
##################### change_score 함수 ####################

def change_score():
    student_id = input('Student ID: ')
    for student in stu_list:
        if student[0] == student_id:
            mid_or_final = input('Mid/Final? ')
            if mid_or_final == "mid":  # 중간고사일 때,
                new_score = int(input("Input new score: "))
                if 0 <= new_score and new_score <= 100:
                    # 바뀌기 이전 score 출력
                    print_table_row()
                    print(student[0] + "\t" + student[1] + "\t" + str(student[2]) + "\t" +
                          str(student[3]) + "\t" + str(student[4]) + "\t" + student[5])

                    student[2] = new_score  # 중간고사 점수 수정
                    student[4] = get_average(
                        student[2], student[3])  # 평균점수 다시 계산
                    student[5] = get_grade(student[4])  # Grade 다시 계산

                    # 바뀐 후 score 출력
                    print("Score changed.")
                    print(student[0] + "\t" + student[1] + "\t" + str(student[2]) + "\t" +
                          str(student[3]) + "\t" + str(student[4]) + "\t" + student[5])

                    # 전체 리스트 다시 정렬
                    stu_list.sort(key=lambda e: e[4], reverse=True)
                    return
                else:
                    return
```

- 먼저, 학번을 입력 받는다. 그러고나서, stu_list 를 for 문으로 돌면서 일치하는 학번을 찾는다. 일치하는 학번이 있다면 다음 단계로 넘어간다. 아니면 return 한다.
- 중간고사 성적을 바꿀 것인지, 기말고사 성적을 바꿀 것인지 물어본다. mid 나 final 로 정확히 입력되면 그 다음 단계로 넘어간다. 아니면 return 한다.

- 몇점으로 수정할 것인지 묻는다. 이때, 0 점과 100 점 사이라면 다음단계로 넘어간다. 아니면 return 한다.
- student 의 바뀌기 이전의 점수를 출력하고, student 에 새로운 중간고사(혹은 기말고사) 점수를 입력한다. 그러고나서, get_average()와 get_grade() 함수를 이용해서, 평균과 학점을 다시 새롭게 계산해준다. 그리고 원래 student 점수와 학점을 수정한다.

e) add()

```
##################### add 함수 ####################


def add():
    existing_id_list = []
    for student in stu_list:
        existing_id_list.append(student[0])  # 이미 존재하는 학번 리스트를 만든다.
    input_id = input("Student ID: ")
    # 이미 학번 존재하면 함수를 return 해버린다.
    if input_id in existing_id_list:
        print("ALREADY EXISTS")
        return
    # 학번이 중복되지 않는 것이 확인되었으므로, 나머지 정보도 입력받는다.
    input_name = input("Name: ")
    input_midterm_score = int(input("Midterm Score: "))
    input_final_score = int(input("Final Score: "))
    # 평균과 학점을 계산한다.
    averge_score = get_average(input_midterm_score, input_final_score)
    grade = get_grade(averge_score)
    # 새로운 학생의 정보 리스트를 만든다.
    new_student_info_list = [
        input_id, input_name, input_midterm_score, input_final_score, averge_score, grade]
    # stu_list에 추가한다.
    stu_list.append(new_student_info_list)
    # 전체 학생 리스트를 평균 순으로 다시 정렬한다.
    stu_list.sort(key=lambda e: e[4], reverse=True)
    print("Student added.")
    return
```

- 이미 존재하는 학번 리스트를 만든다.
- 학번을 입력받는다.
- 이미 학번이 존재하면 함수를 return 한다.
- 학번이 중복되지 않는 것이 확인되었으면, 중간고사 점수와 기말고사 점수를 입력 받는다.

- 입력 받은 중간고사 점수와 기말고사 점수를 가지고 평균점수와 학점을 계산한다.
- new_student_info_list 에 앞서 구한 정보들까지 포함하여, 학번, 이름, 중간점수, 기말점수, 평균점수, 학점을 넣는다.
- stu_list 에 new_student_info_list 를 추가한다.
- 평균을 기준으로 내림차순으로 정렬한다.
- "Student added."를 출력하고 함수를 끝낸다.(return)

f) search_grade()

```
##################### searchgrade 함수 ####################


def search_grade():
    existing_grade_list = []
    for student in stu_list:
        if student[5] not in existing_grade_list:
            existing_grade_list.append(student[5])  # 이미 존재하는 학번 리스트를 중복없이 만든다.
    # 찾고있는 학점을 입력받는다.
    grade_to_search = input('Grade to search: ')
    # "A", "B", "C", "D", "F" 중 하나가 아니라면, return 한다.
    if grade_to_search not in ["A", "B", "C", "D", "F"]:
        return
    # 학생들의 학점 중에 없다면, return 한다.
    elif grade_to_search not in existing_grade_list:
        print("NO RESULTS.")
        return
    else:
        searching_grade_student_list = []
        for student in stu_list:
            if student[5] == grade_to_search:
                searching_grade_student_list.append(student)
        for student in searching_grade_student_list:
            print(student[0] + "\t" + student[1] + "\t" + str(student[2]) + "\t" +
                  str(student[3]) + "\t" + str(student[4]) + "\t" + student[5])
        return
    return
```

- 이미 존재하는 학점 리스트를 중복없이 만든다.
- 찾고 있는 학점을 입력받는다.
- 만약, 찾으려는 학점이 A,B,C,D,F 중에 없다면 return 한다.
- elif, 찾으려는 학점이 존재하지 않는다면, "NO RESULTS."를 출력한다.
- else, searching_grade_student_list 를 만들고, for 문을 돌면서 해당 학점을 가진 학생들을 위 리스트에 append 한다.

- searching_grade_student_list 에 존재하는 학점들을 한 명씩 정보를 출력한다.

g) remove()

```
##################### remove 함수 #####################


def remove():
    # 목록에 아무도 없을 경우, return 한다.
    if len(stu_list) == 0:
        print("List is empty.")
        return

    # 학번 리스트를 만든다.
    existing_id_list = []
    for student in stu_list:
        existing_id_list.append(student[0])

    # 학번을 입력받는다.
    input_id = input("Student ID: ")
    if input_id not in existing_id_list:
        print("NO SUCH PERSON.")
        return
    else:
        for student in stu_list:
            if student[0] == input_id:
                stu_list.remove(student)
                break
        print("Student removed.")
        return
```

- 목록에 아무도 없을 경우, "List is empty."를 출력하고 return 한다.
- 이미 존재하는 학번만 담은 리스트를 for 문을 돌면서 만든다.
- 학번을 입력받는다.
- 학번이 existing_id_list 에 존재하지 않으면, 해당 학번을 가진 학생이 없는 것이므로, "NO SUCH PERSON."을 출력한다.
- 아니면, 해당 학번을 가진 학생을 stu_list 에서 remove 하고, "Student removed."를 출력한다.

h) quit()

```
##################### quit 함수 #####################


def quit():
    save_data_yes_or_no = input("Save data?[yes/no] ")
    if save_data_yes_or_no == "yes":
        file_name_to_write = input("File name: ")
        fw = open(f"./{file_name_to_write}", "w")
        for student in stu_list:
            data = student[0] + "\t" + student[1] + "\t" + str(student[2]) + "\t" + str(
                student[3]) + "\t" + str(student[4]) + "\t" + student[5] + "\n"
            fw.write(data)
        fw.close()
    return
```

- 파일을 저장할지 여부를 yes/no 로 입력받는다.
- yes 일 때는, 파일명을 입력받고, 해당 데이터를 텍스트 파일에 써서 저장한다.
- no 일 때는 파일을 저장하지 않고 함수를 return 한다.

i) readline_and_append()

```
##################### 파일을 한줄씩 읽어서 리스트에 추가하는 함수 #####################


def readline_and_append(readed_file):
    for line in readed_file:
        first_name = line.split()[2].lower()  # "gildong"
        full_name = line.split()[1] + " " + \
            line.split()[2]  # "Hong Gildong"
        stu_number_name_mid_final_list = [line.split()[0], full_name, int(line.split()[3]), int(line.split()[
            4])]  # ["20180001","Hong Gildong", 84, 73]
        # gildong = ["20180001","Hong Gildong", 84, 73] 형태로도 저장됨
        globals()[first_name] = stu_number_name_mid_final_list
        # 중첩 리스트 형식으로도 저장
        stu_list.append(stu_number_name_mid_final_list)
    return
```

- 텍스트 파일을 읽어온 후, stu_list 를 생성한다. 그리고 사람 이름을 변수명으로 갖는 변수를 생성한다.(하지만 이후 사용하지는 않음)

j) get_average()

```
##################### 평균구하기 함수 #####################


def get_average(mid, final):
    return (mid + final)/2
```

- 중간고사 점수와 기말고사 점수를 합쳐서 평균을 구해준다.

k) get_grade()

```
###################### 학점구하기 함수 ######################


def get_grade(score):
    if score >= 90:
        return 'A'
    elif score >= 80 and score < 90:
        return 'B'
    elif score >= 70 and score < 80:
        return 'C'
    elif score >= 60 and score < 70:
        return 'D'
    else:
        return "F"
```

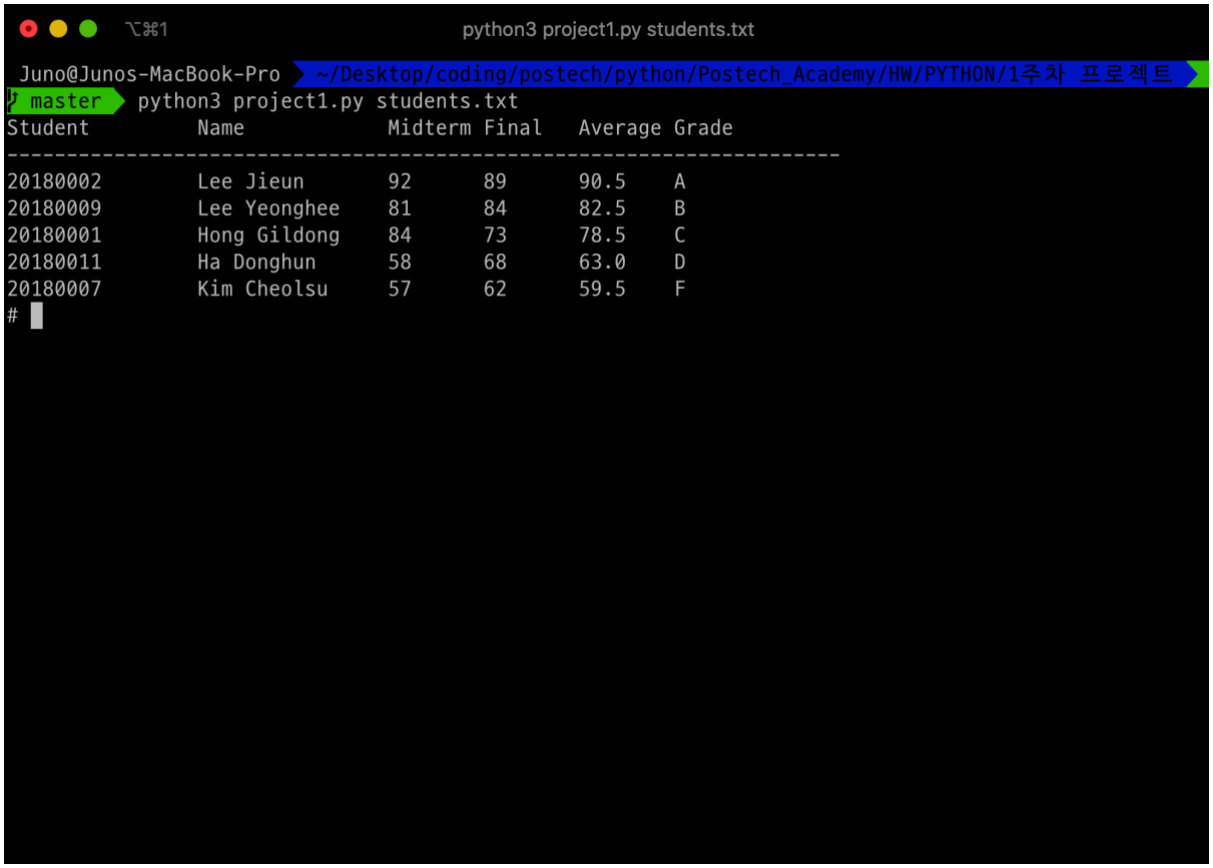
- 평균 점수를 가지고 학점을 계산해준다.

l) print_table_row()

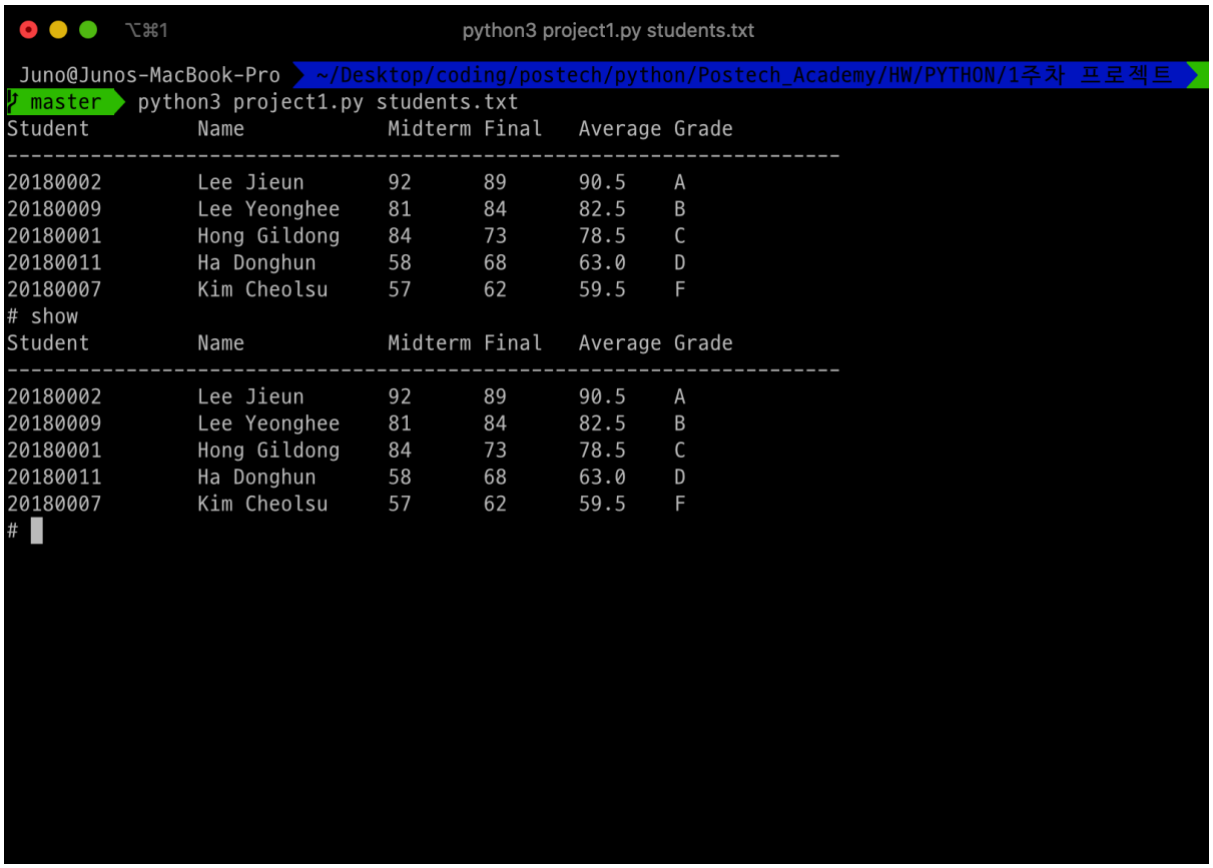
```
###################### 표 윗 부분 출력해주는 함수 ######################


def print_table_row():
    print("Student" + "\t\t" + "Name" + "\t\t" + "Midterm" + "\t" +
          "Final" + "\t" + "Average" + "\t" + "Grade")
    print("-" * 70)
    return
```

- 표를 출력할 때, 맨 윗부분을 출력해준다.(행 이름)

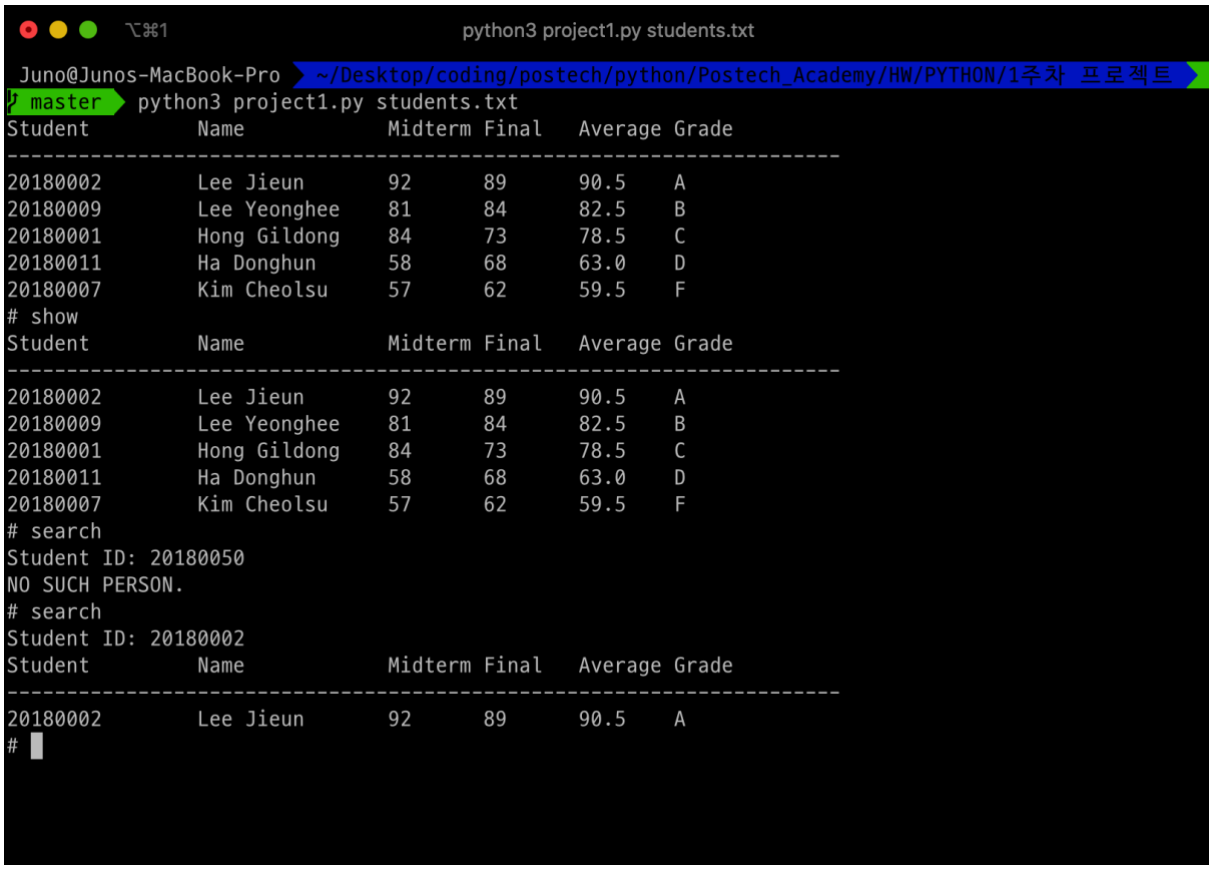
4. 프로그램 실행방법 및 예제



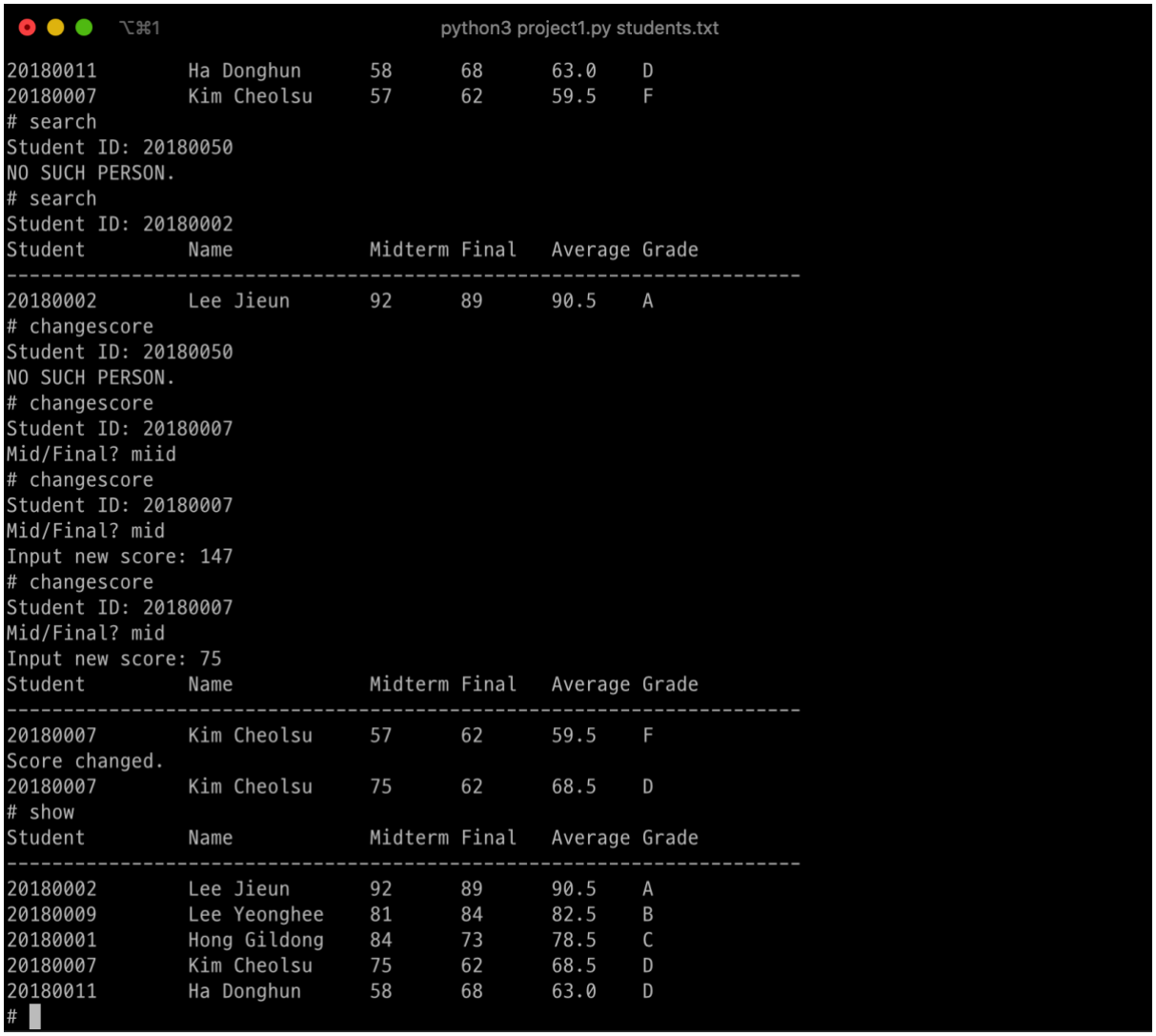
: 프로그램을 실행하면서, 경로 내에 있는 students.txt 파일을 불러온다.



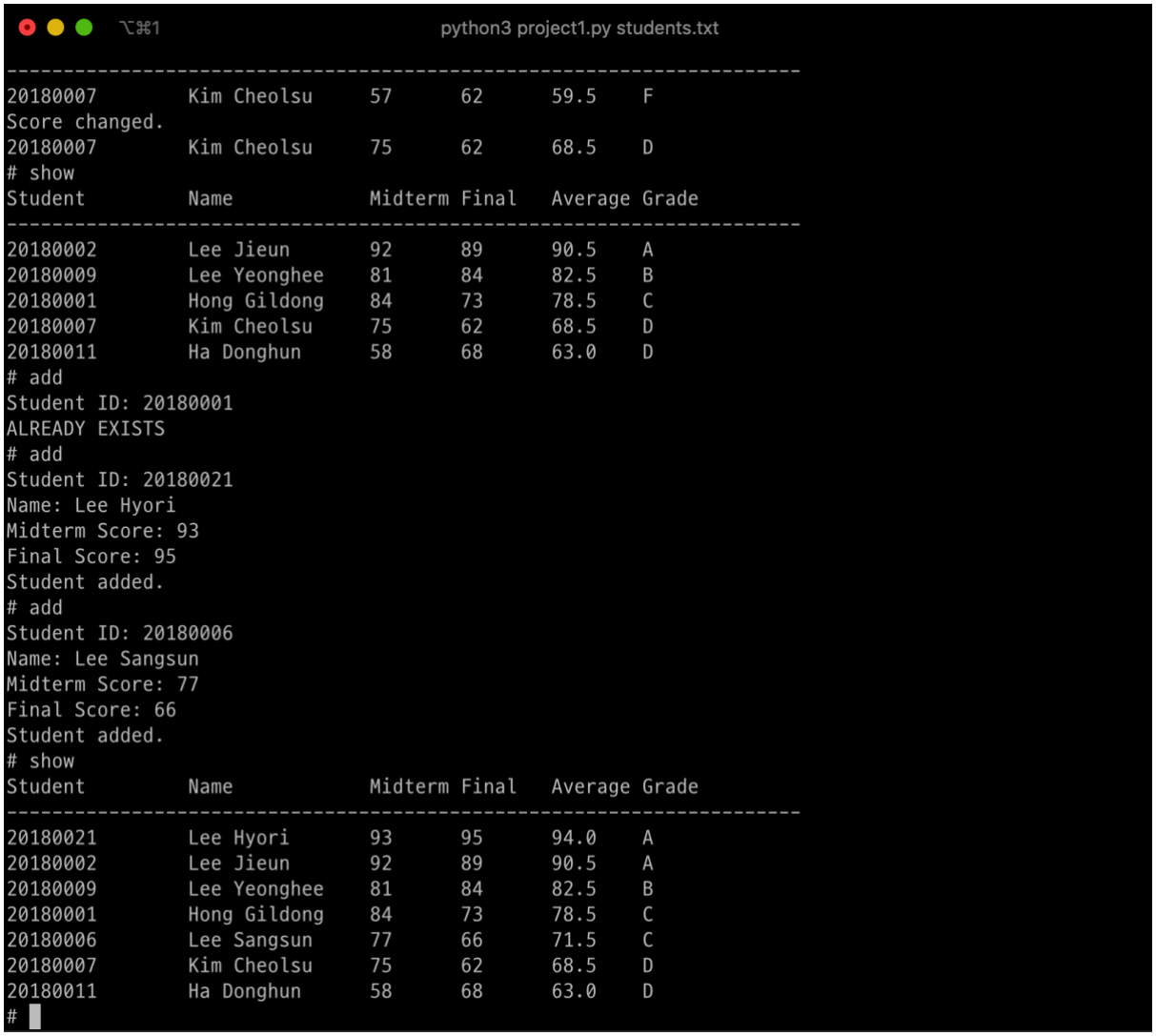
: show 명령어를 입력하면, 현재 목록에 있는 모든 학생들을 테이블로 보여준다.



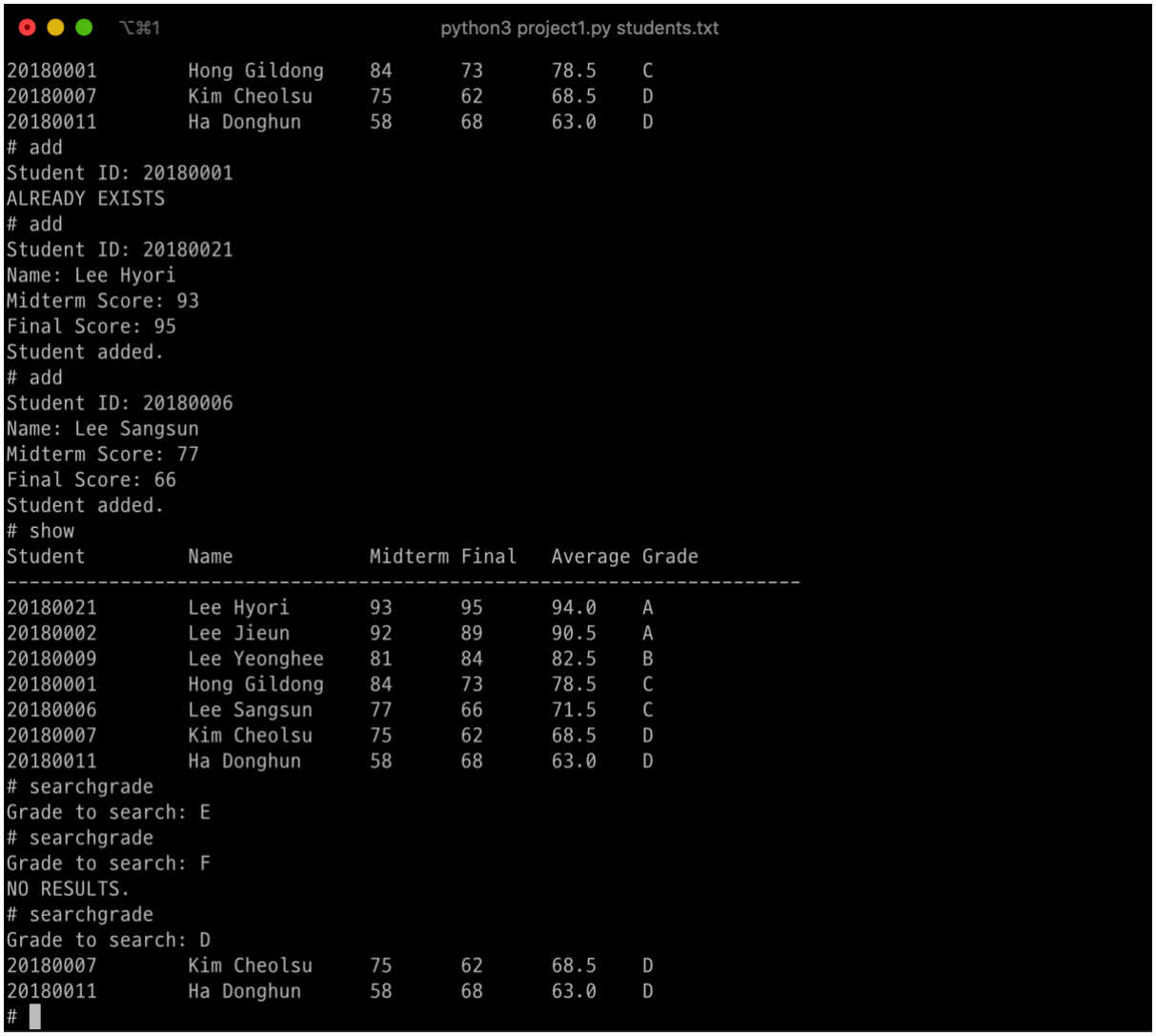
: search 명령어를 입력하고, 존재하는 학번을 입력하면, 해당하는 학생의 정보를 출력한다. 존재하지 않는 학번을 입력하면, "NO SUCH PERSON"을 출력한다.



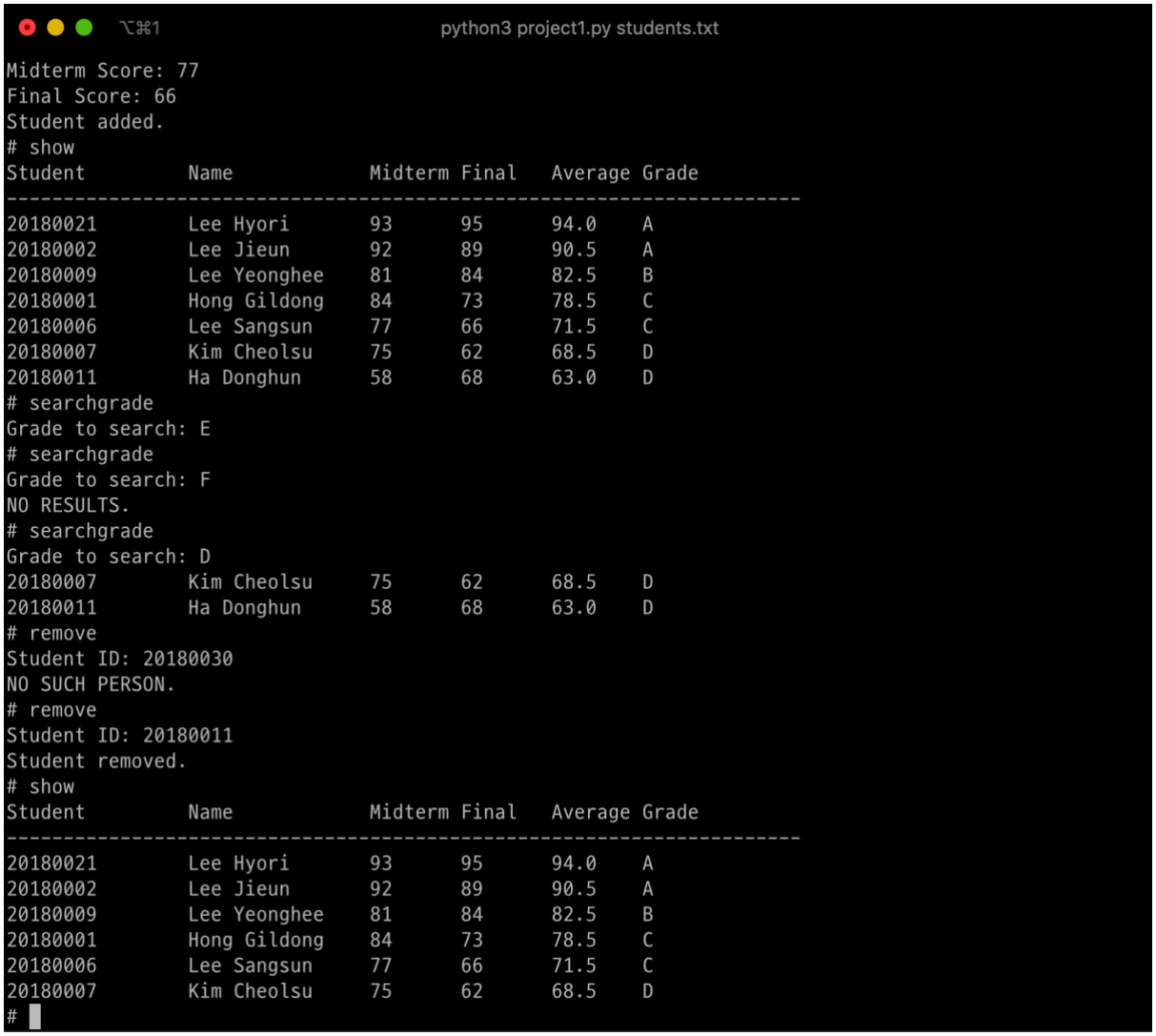
: changescore 를 입력하면, 학번을 입력받는다. 학번이 존재하고, mid/final 중 알맞은 선택을 하고, 0~100 점 사이의 점수를 입력하면, 해당 학생의 점수를 수정한다. 그리고 수정되기 전, 수정된 후의 점수를 출력한다.



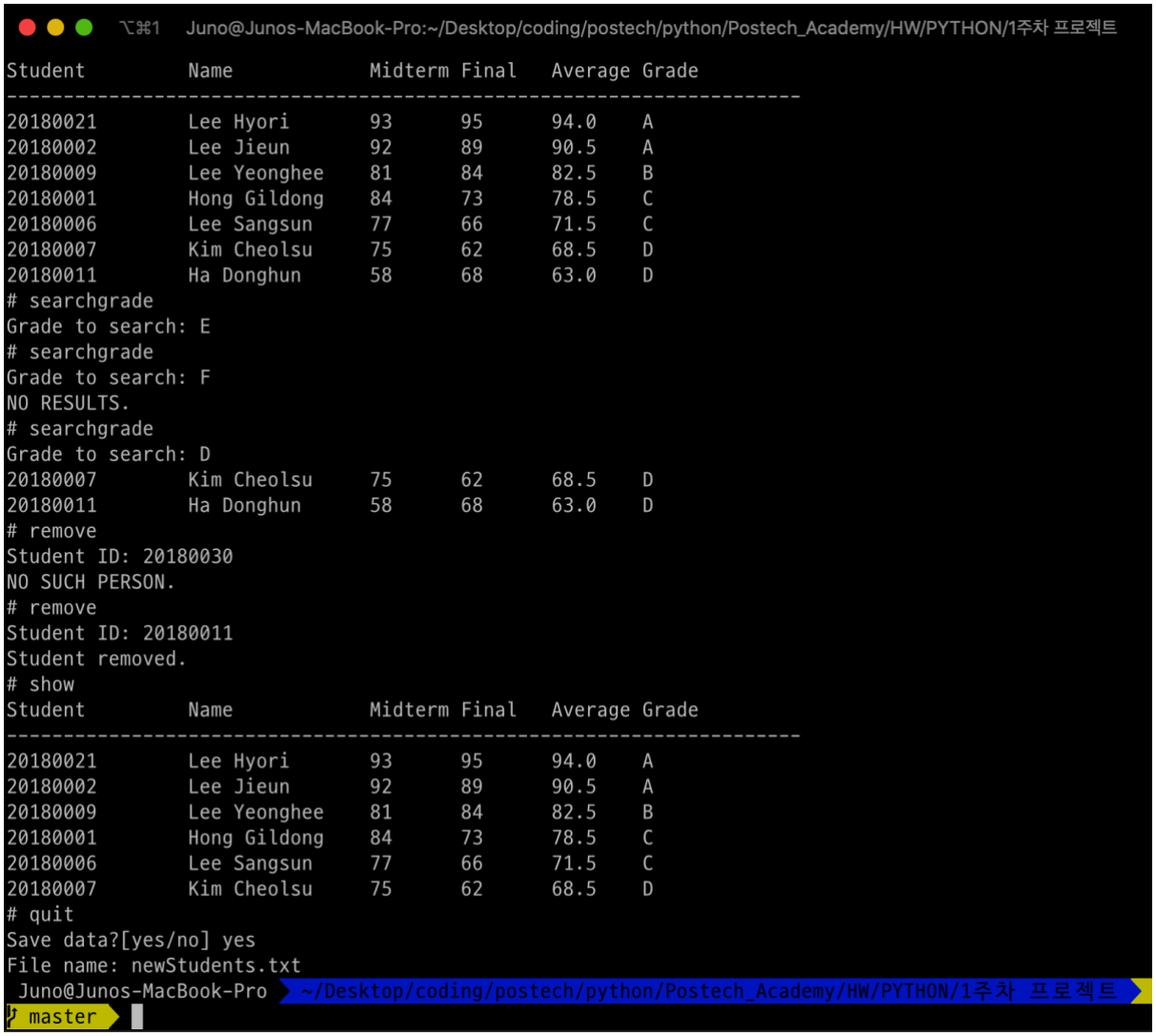
: add 명령어를 통해 새로운 학생을 등록할 수 있다. 현재 존재하지 않는 학번을 입력하면, 이름, 중간점수, 기말점수를 물어보고 내부적으로 평균과 학점을 계산하여 목록에 추가한다. 마지막으로 show 를 통해 성공적으로 등록됐는지 여부를 확인한 화면이다.



: searchgrade 명령어를 통해 해당 학점을 가진 학생들을 출력한다. 알맞은 형태의 학점이 아니면, 다시 명령어 입력을 요청하고, 해당 학점을 가진 학생이 한 명도 없다면, NO RESULTS.를 출력한다. 해당 학점을 가진 학생이 있다면, 해당 학점을 가진 모든 학생들의 정보를 출력한다.



: remove 명령어를 통해 학생 정보를 삭제한다. 학번을 입력받아 삭제하고, 입력받은 학번에 해당하는 학생이 없으면, NO SUCH PERSON.을 출력한다. 아래는 정상적으로 삭제되었는지 확인하는 과정이다.



: quit 명령어를 통해 프로그램을 나간다. 나갈 때, data 저장 여부를 물어보며, yes 를 선택하면, 생성하고 싶은 파일의 파일명을 입력받아 생성한다.

5. 토론

- 새로운 정보를 입력받거나, 수정이 된 후에 평균점수 순으로 정렬을 한 다음에 저장을 하는 형태로 프로그래밍을 하였는데, 매번 하는 것이 아니라, 데이터를 최종적으로 저장할 때만 하는 것이 좋을까 고민을 해봐야겠다.

- Input() 함수가 아닌, sys.stdin.readline 함수가 더 효율적이라는 것을 검색결과 알아낼 수 있었는데, 이 부분에 대해 더 알아보아야겠다.

6. 결론

이번 프로그램을 만들면서 가장 크게 배운 점은, 다양한 기능들을 하는 프로그램을 제작할 때는, 계층에 맞게 객체화를 잘해야한다는 것이었다. 클래스와 함수를 잘 활용하여 재사용이 가능하도록 코딩을 하는 능력을 길러야겠다는 생각을 했다.

7. 개선방향

- 학생을 추가할 때, 학생의 점수를 수정할 때와 마찬가지로 중간고사와 기말고사 점수 범위를 제한해놓아 잘못된 데이터가 입력받지 않도록 하면 좋을 것 같다.
- 학생 정보를 이중 리스트로 관리하는 것보다는, Student 클래스를 만들고, 객체를 하나씩 생성하여 관리하면 더 깔끔한 프로그램이 됐을 것 같다.