SMART STUDENT EVALUATION SYSTEM

Nimash Dilanka

PYTHON ONLINE CLASS – version 2

Contents

1 Introduction

A school need a system on keep annual grades and achievements of each student as a digital information. End goal is to automatically evaluate the improvement of each student nad notify to responsible parties. As the first version, teachers would like to see a system that can store students details. This is a CMD alpplication (command line interface).

2 Main screen

After initiation, software shows following message to user (either a teacher or student)

```
**********************************************************************************************
**********************************************************************************************

*****************************Welcome to Smart student evaluation System*****************************

**********************************************************************************************

**********************************************************************************************
```

3 Main Commands

3.1 add_student

3.1.1 syntax

add_student <student_name> <DOB>

3.1.2 operations

keep a note about the student name and Date of Birth until application terminates. Two students from same name can not exist. Student name should be single word with only letters(case-sentitive). Everytime a student is added into the system a internal id is generated by system and keep that with student data.

3.1.3 Error messages

3.1.3.1 Two students can not have same name

3.1.3.2 Student name have should have no white spaces

3.1.3.3 Student name should not have numbers included

3.1.3.4 DOB should match with DD/MM/YYYY format

3.1.3.5 Invalid syntax press 'help –' to check valid command list

3.2 modify_student

3.2.1 syntax

modify_student <old_student_name> <new_student_name> <DOB>

3.2.2 operations

modify student DOB and name.

3.2.3 Error messages

3.2.3.1 Two students can not have same name

3.2.3.2 Student name have should have no white spaces

3.2.3.3 Student name should not have numbers included

3.2.3.4 Student with <student_name> does not exist

3.2.3.5 DOB should match with DD/MM/YYYY format

3.2.3.6 Invalid syntax press 'help –' to check valid command list

3.3 remove_student

3.3.1 syntax

remove_student <student_name>

3.3.2 operations

remove a student from application. System should not allow to remove student if there is scores stored for this student. If deletion is successful, id of this student wont be assigned to any new student until application terminates.

3.3.3 Error messages

3.3.3.1 Student with <student_name> does not exist

3.3.3.2 Student name should have no white spaces

3.3.3.3 Student name should not have numbers included

3.3.3.4 Can not be deleted. Student has scores stored in the system

3.3.3.5 Invalid syntax press 'help –' to check valid command list

3.4 help_student

3.4.1 syntax

help_student <student_name>

3.4.2 operations

show all details of the student.

3.4.3 Error messages

3.4.3.1 Student with <student_name> does not exist

3.4.3.2 Invalid syntax press 'help –' to check valid command list

3.5 add_subject

3.5.1 syntax

add_subject <name> <stream>

3.5.2 operations

add subject name and stream (O/L or A/L) into the system. Subject is a single word with only letters(case-sencitive). Everytime a subject is added into the system a internal id is generated by system and keep that with subject data.

3.5.3 Error messages

3.5.3.1 Two subjects can not have same name

3.5.3.2 Subject name have should have no white spaces

3.5.3.3 Student name should not have numbers included

3.5.3.4 Stream should be either A/L or O/L

3.5.3.5 Invalid syntax press 'help –' to check valid command list

3.6 modify_subject

3.6.1 syntax

modify_subject <old_ subject _name> <new_ subject _name> <stream>

3.6.2 operations

modify subject stream and name.

3.6.3 Error messages

3.6.3.1 Two subjects can not have same name

3.6.3.2 Subject name should have no white spaces

3.6.3.3 Subject name should not have numbers included

3.6.3.4 Subject with <subject_name> does not exist

3.6.3.5 Stream should be either A/L or O/L

3.6.3.6 Invalid syntax press 'help –' to check valid command list

3.7 remove_subject

3.7.1 syntax

remove_subject <subject_name>

3.7.2 operations

remove a subject from application. System should not allow to remove subject if there is scores stored for this subject. If deletion is successful, id of this subject wont be assigned to any new subject until application terminates.

3.7.3 Error messages

3.7.3.1 Subject with <subject_name> does not exist

3.7.3.2 Can not be deleted. Subject has scores stored in the system

3.7.3.3 Invalid syntax press 'help –' to check valid command list

3.8 show_subject

3.8.1 syntax

show_subject <subject_name>

3.8.2 operations

show all details of the subject.

3.8.3 Error messages

3.8.3.1 Subject with <subject_name> does not exist

3.8.3.2 Invalid syntax press 'help –' to check valid command list

3.9 add_mark

3.9.1 syntax

add_mark <student_name> <subject_name> <score>

3.9.2 operations

Keep a note about the student's score until application terminates.

3.9.3 Error messages

3.9.3.1 Student with <student_name> does not exist

3.9.3.2 Subject with <subject_name> does not exist

3.9.3.3 This student has marks for this subject already

3.9.3.4 Student mark should be a number

3.9.3.5 Student mark should be greater than zero

3.9.3.6 Student mark should be less or equal to 100

3.9.3.7 Invalid syntax press 'help –' to check valid command list

3.10 modify_mark

3.10.1 syntax

modify_mark <old_student_name> <old_subject_name> <new_student_name> <new_subject_name> <score>

3.10.2 operations

Update student name, subject name and score.

3.10.3 Error messages

3.10.3.1 Student with <student_name> does not exist

3.10.3.2 Subject with <subject_name> does not exist

3.10.3.3 This student has no marks for this subject yet

3.10.3.4 Student mark should be a number

3.10.3.5 Student mark should be greater than zero

3.10.3.6 Student mark should be less or equal to 100

3.10.3.7 Invalid syntax press 'help –' to check valid command list

3.10.3.7.1

3.11 remove_mark

3.11.1 syntax

remove_mark <student_name> <subject_name>

3.11.2 operations

Clear student's score. After clearing score , user can add it again.

3.11.3 Error messages

3.11.3.1 Student <student_name> has no score to clear

3.11.3.2 Student name have should have no white spaces

3.11.3.3 Student name should not have numbers included

3.11.3.4 Invalid syntax press 'help –' to check valid command list

3.12 show_mark

3.12.1 syntax

show_mark <student_name> <subject_name>

3.12.2 operations

show all details of the score.

3.12.3 Error messages

3.12.3.1 Subject with <subject_name> does not exist

3.12.3.2 Student with <student_name> does not exist

3.12.3.3 Invalid syntax press 'help –' to check valid command list

3.13 show_all_marks

3.13.1 syntax

show_all_marks <student_name>

3.13.2 operations

Show all marks related to the student as a table.

3.13.3 Error messages

3.13.3.1 Student with <student_name> does not exist

3.13.3.2 Invalid syntax press 'help –' to check valid command list

3.14 show_all_students

3.14.1 syntax

show_all_students <subject_name>

3.14.2 operations

Show all students related to the subject as a table.

3.14.3 Error messages

3.14.3.1 Subject with <subject_name> does not exist

3.14.3.2 Invalid syntax press 'help –' to check valid command list

3.15 help

3.15.1 syntax

help

3.15.2 operations

Show syntax, operation description, details of each error message of all the available commands.

3.15.3 Error messages

3.15.3.1 Invalid syntax press 'help –' to check valid command list

3.16 exit

3.16.1 syntax

exit

3.16.2 operations

Exit the application showing following message.

```
**********************************************************************************************
**********************************************************************************************

**********************************Good Bye !! Have a nice day**********************************

**********************************************************************************************

**********************************************************************************************
```

3.16.3 Error messages

3.16.3.1 Invalid syntax press 'help –' to check valid command list

4 Data Entities

4.1 Subject Entity

All students learn subjects like “English” etc. The subject is a collection of 3 properties.

Subject
Id
SubjectName
Stream

4.1.1 Id

This is a number which generated by system. (not given by user). Each subject has its own id. No same id will be owned by different subjects anytime. For Ex: imagine “Sinhala” language has id 3 in the system. Then the Subject “Sinhala” is removed from system. Even after that moment, id 3 will not be assigned to any of new Sunjects.

4.1.2 SubjectName

The name of the subject. No two sunjects should have same name(case-sensitive).

4.1.3 Stream

Stream is either “A/L” or “O/L”. no other value is permitted.

4.1.4 Promoted data type

Use a list of dictionaries to remember a subject. One dictionary is a one subject. Create a separate .py file called SubjectDataCenter.py for all subject entity related functions. Those functions are:

4.1.4.1 Add a subject

SubjectDataCenter.py - C:/Users/nimas/Desktop/SubjectDataCenter.py (3.9.0)

File Edit Format Run Options Window Help

```
#here stores all the subjects
subjectEntities=[]

def addSubject(id, subjectName, stream):
    "store the subject inside the subject data list. return True if operation is successful"
    #todo- check the a subject with this id already exist in system. If so, return false
    #todo- check the a subject with this subject name already exist in system. If so, return false
    #return False;

    #add the new Subject
    newSubject={}
    newSubject["id"]=id
    newSubject["subjectName"]=subjectName
    newSubject["stream"]=stream
    subjectEntities.add(newSubject)
    return True;
```

4.1.4.2 Modify a subject

```
def modifyStudent(id, name, dob):
    "modify content of a already stored Student. return True if operation is successful"
    #todo- check the a Student with this id already exist in system. If not, return false
    #return False;

    #choose the Student
    for student in studentEntities:
        if(student["id"]==id)
            selectedStudent=student

    selectedStudent["name"]=name
    selectedStudent["dob"]=dob
    return True;
```

4.1.4.3 Delete a subject

```
def deleteSubject(id):
    "delete a subject from the system. return Tue if operation is successful"
    #todo- check the a subject with this id already exist in system. If not, return false
    #return False;

    #choose the Subject
    for subject in subjectEntities:
        if(subject["id"]==id)
            selectedSubject=subject
    subjectEntities.remove(selectedSubject)
    return True;
```

4.1.4.4 Get subject by id

```
def getSubjectById(id):
    "return the subjct that has given id. Otherwise return None"

    #choose the Subject
    for subject in subjectEntities:
        if(subject["id"]==id)
            return subject.copy(); # give a copy of the dictionary as returned value.
    return None;
    #we can check this using if statement 'if ret_value is not None'
```

4.1.4.5 Get subject by subject name

```
def getSubjectBySubjectName(subjectName):
    "return the subjct that has given id. Otherwise return None"

    #choose the Subject
    for subject in subjectEntities:
        if(subject["subjectName"]==subjectName)
            return subject.copy(); # give a copy of the dictionary as returned value.
    return None;
    #we can check this using if statement 'if ret_value is not None'
```

4.2 Student Entity

Each student is a collection of 3 properties.

Student
Id
Name
DOB

4.2.1 Id

This is a number which generated by system. (not given by user). Each student has its own id. No same id will be owned by different students anytime. For Ex: imagine student “Kasun” has id 3 in the system. Then the student “Kasun” is removed from system. Even after that moment, id 3 will not be assigned to any of new students.

4.2.2 Name

The name of the student. A Single string with letters only. No spaces allowed.

4.2.3 DOB

Date of Birth is given in “DD/MM/YYYY” format as a single string. For ex: 23 March 2020 is 23/03/2020.

4.2.4 Promoted data type

Use a list of dictionaries to remember a subject. One dictionary is a one subject. Create a separate .py file called StudentDataCenter.py for all student entity related functions. Those functions are:

4.2.4.1 Add a student

StudentDataCenter.py - C:/Users/nimas/Desktop/StudentDataCenter.py (3.9.0)

File Edit Format Run Options Window Help

```
#here stores all the students
studentEntities=[]

def addStudent(id, name, dob):
    "store the student inside the student data list. return True if operation is successful"
    #todo- check the a Student with this id already exist in system. If so, return false
    #todo- check the a Student with this Student name already exist in system. If so, return false
    #todo- check the DOB is correct. If not, return false
    #return False;

    #add the new Student
    newStudent={}
    newStudent["id"]=id
    newStudent["name"]=StudentName
    newStudent["dob"]=dob
    studentEntities.add(newStudent)
    return True;
```

4.2.4.2 Modify a student

```
def modifyStudent(id, name, dob):
    "modify content of a already stored Student. return True if operation is successful"
    #todo- check the a Student with this id already exist in system. If not, return false
    #return False;

    #choose the Student
    for student in studentEntities:
        if(student["id"]==id)
            selectedStudent=student

    selectedStudent["name"]=name
    selectedStudent["dob"]=dob
    return True;
```

4.2.4.3 Delete a student

```
def deleteStudent(id):
    "delete a Student from the system. return True if operation is successful"
    #todo- check the a Student with this id already exist in system. If not, return false
    #return False;

    #choose the Student
    for student in studentEntities:
        if(student["id"]==id)
            selectedStudent=student
    studentEntities.remove(selectedStudent)
    return True;
```

4.2.4.4 Get student by id

```
def getStudentById(id):
    "return the student that has given id. Otherwise return None"

    #choose the Student
    for student in studentEntities:
        if(student["id"]==id)
            return student.copy(); # give a copy of the dictionary as returned value.
    return None;
    #we can check this using if statement 'if ret_value is not None'
```

4.2.4.5 Get student by name

```
def getStudentByName(name):
    "return the student that has given name. Otherwise return None"

    #choose the Student
    for student in studentEntities:
        if(student["studentName"]==studentName)
            return student.copy(); # give a copy of the dictionary as returned value.
    return None;
    #we can check this using if statement 'if ret_value is not None'
```

4.3 Student Subject relationship

This has score for each student per each subject.

Student_Subject
id
Subject_id
Student_id
Score

4.3.1 Id

This is a number which generated by system. (not given by user). Each relationship has its own id

4.3.2 Student_id

This is id of the student which the score is belong to.

4.3.3 Subject_id

This is the id of the subject of the student.

4.3.4 Score

This is the score for given subject for given student.

4.3.5 Promoted data type

Use a list of dictionaries to remember a subject. One dictionary is a one relationship entity. Create a separate .py file called StudentSubjectRelation.py for all relationship entity related functions. Those functions are:

4.3.5.1 Add student subject relation

StudentSubjectRelation.py - C:/Users/nimas/Desktop/StudentSubjectRelation.py (3.9.0)

File Edit Format Run Options Window Help

```
#here stores all the relationship between student and score
studentSubjectRelations=[]

def addStudentSubjectRelation(id, studentId, subjectId, score):
    #todo- check the a relationship with this id already exist in system. If so, return false
    #todo- check the id, studentId, subjectId already exist in system. If so, return false
    #return False;

    #add the new StudentSubjectRelation
    newStudentSubjectRelation={}
    newStudentSubjectRelation["id"]=id
    newStudentSubjectRelation["studentId"]=studentId
    newStudentSubjectRelation["subjectId"]=subjectId
    newStudentSubjectRelation["score"]=score
    studentSubjectRelations.add(newStudentSubjectRelation)
    return True;
```

4.3.5.2 Modify student subject relation

```
def modifyStudentSubjectRelation(id, studentId, subjectId, score):
    #todo- check the a relationship with this id already exist in system. If not, return false
    #return False;

    #choose the StudentSubjectRelation
    for relationship in studentSubjectRelations:
        if(relationship["id"]==id)
            selectedRelationship=relationship
    selectedRelationship["studentId"]=studentId
    selectedRelationship["subjectId"]=subjectId
    selectedRelationship["score"]=score
    return True;
```

4.3.5.3 Delete student subject relation

```
def deleteStudentSubjectRelation(id):
    "delete a Relationship from the system. return True if operation is successful"
    #todo- check the a relationship with this id already exist in system. If not, return false
    #return False;

    #choose the StudentSubjectRelation
    for relationship in studentSubjectRelations:
        if(relationship["id"]==id)
            selectedRelationship=relationship
    studentSubjectRelations.remove(selectedRelationship)
    return True;
```

4.3.5.4 Get all relationships by student id

```
def getRelationshipsByStudentId(studentId):
    "return the relationship list which has mentioned studentId"

    selectedRelationships=[]
     for relationship in studentSubjectRelations:
        if(relationship["studentId"]==subjectId)
            selectedRelationships.add(relationship.copy())
    return selectedRelationships;
```

4.3.5.5 Get all relationships by subject id

```
def getRelationshipsBySubjectId(subjectId):
    "return the relationship list which has mentioned subjectId"

    selectedRelationships=[]
     for relationship in studentSubjectRelations:
        if(relationship["subjectId"]==subjectId)
            selectedRelationships.add(relationship.copy())
    return selectedRelationships;
```

4.4 Example use

This is provided as .py files