The main file for your project file should be call StudentManagement.py. This is where you will write your general program structures and call other classes from. It will be the main prompt. All of your logic code should be in a main() method.

Each students record should fill a single line in the file. The students record should follow the below format in the text file:

fodselsNummer,firstName,lastName,age,email,programmingCourse

An example of this format and data is included in the provided text file *StudentRecords.txt .*

To create and store the student you should implement a **Student class** called *StudentRecordClass* . The *StudentRecordClass should be able to store the following information:*

- *fodselsNummer*
- *firstName*
- *lastName*
- *age*
- *email*
- *programmingCourse*

It should also have a function called *DisplayName().* The *DisplayName* function should print the students full name i.e. First Name (Peter) Last name (Pan) should print out: Peter Pan.

The *StudentRecordClass* class should be saved in its own .py file.

program should start by prompting you with the following question:
"Would you like to enter a student's information? Type Y for Yes or N for No:".

- If the user types *Y,* you should start the process to enter information about the student.
- If the user types *N,* you should execute the code for Part 2 of the assessment.

Example of program start Output:

```
Would you like to enter a student's information? Type Y for Yes or N for No :
```

In brief the program should continue prompting you for student information until the user types *N*.

When the program is executed after choosing option Y, there should be sequent prompting for all the student's information. This information should be added to an instance of the *StudentRecordClass* class. Once you have typed all the information needed for one student you should be able to call a function to display the message
"Writing student information to Student Record File"
and write that student's information to the *StudentRecords.txt* file in the correct format. Make sure all storage of programming choice entries is done in lowercase text.

Example of student information capture process:

```
Would you like to enter a student's information? Type Y for Yes or N for No : Y
Fodselsnummer: 11056027043
First name: John
Last name: Smith
Age:17
E-mail johnnyboy@yahoo.com
Programming Course: python
Writing student information to Student Record File
```

IMPORTANT:
You should implement general error catching in your code which states the following for all input errors:

"Oops something is buggy"

Output example:

```
Oops something is buggy
```

It should also have more specific (and appropriate) handling for Value Errors. If the user inputs an incorrect value, they should be re-prompted for the correct value type and then the program should continue.

Part 2

This code should be implemented when the user types N, to indicate the do not want to add another student's information. The calls for this code will continue to work with the StudentRecords.txt file.

This section will involve asking the user what else they would like to do. Based on their decision you will need to call specific appropriate functions. The instructions will first present the questions to be asked. Then it will explain the functions.

The program should display the prompts in the following order.

```
1.Would you like to see a list of all registered students?
2.Would you like to see a class list for a specific subject?
3.Would you like to see who your oldest student is?
4.Would you like to see who your youngest student is?
Enter the number for the selected task, or X to skip this:
```

Based on the number execute the appropriate code and functions. If the user enters an **uppercase or lowercase** X then run the code from Part 3.

Write code for the following functions:

1. `DisplayAllStudents()`

   This function should retrieve all the student's information from the `StudentRecords.txt` file. As each student is retrieved, they should be stored as an instance `StudentRecordClass` object. The whole collection should be stored as a list of student objects. Once you have the complete list you should print the list to show the full name of each student.

   Output should be:

   The students registered are:
   Fullname1
   Fullname 2
   Fullname 3

   Example Output:

```
The students registered are:
John Smith
Jane Doe
Peter Pan
```

2. `DisplaySubjectClassList()`

   This function must accept a *subjectname* as a parameter i.e. Python/Java/C/Php/Ruby. Based on this name it should then go retrieve from the file a list of students taking the subject. As each student is retrieved, they should be stored as an instance `StudentRecordClass` object. The whole collection should be stored as a list of `StudentRecordClass` objects. Once you have the complete list you should print the list to show the full name of each student.

   Output should be:

   The students registered for *subjectname* are:
   Fullname 1
   Fullname 2
   Fullname 3

   Example Output:

```
The students registered for Python are:
John Smith
Lizzy Fizz
Catherine Tabby
```

3. *DisplayOldest()*

   This function should retrieve the list of students and print the full name of the oldest student. The function should be able to print multiple names if several students are the oldest.

   Example Output:

```
The eldest student is Peter Pan. He is 65.
```

4. *DisplayYoungest()*

   This function should retrieve the list of students determine who has the lowest age and print their full name. The function should be able to print multiple names if there are several students the same age and it is the youngest age.

   Example Output:

```
The youngest student(s) is/are Lizzy Fizz, John Smith, Baby Doe. Age 17.
```

Part 3

This code should be implemented when the user types *x* or *X*. The code requires the creation of a separate class. Call the instance of this new class within StudentManagement.py as necessary.

This code should be created in a separate `EncodeDecodeClass`. In a separate .py file called `EncodeDecodeClass.py`.

This code will ask the user if they want to create an encrypted version of the file. This will be a *Y/N* question.

- If the user enters *Y*, implement the code discussed in next paragraph.
- If the user enters *N* print "Would you like to decode the encoded file? Y/N"
  - If they type *Y* Implement Part 4's code
  - Otherwise print:
    "The assessment is over. Have a nice day."

Consider the following if the user typed *Y*:
Let's assume we now have a *StudentRecords.txt* file full of information. Now we want to share that information with lecturers who have appropriate rights to see that information. But we are concerned about the security of sharing the file and the need to keep student information private. As such, we want to encode the file into a separate encoded file copy.

encode the file using a simple encoding method. A Shift Cipher should be used in this assessment. However, the encoding implementation must **NOT** be ROT13. Include a Docstring above this method explaining in 30 to 100 words how a Shift Cipher works.

Write a function call *EncodeStudentList()*. The function should be used for encoding the personal information in the file.

- The function call should receive a filename to work with, as a parameter.

- This method should implement a ROT(n), where *n* is the integer 5. So, the alphabetic characters should be shifted 5 letters in the alphabet.
- For all numeric information in the file you should just swop the first and the last digit. e.g 17051630602 becomes 27051630601.
- Do not alter the commas.

As you encode each record, write the now encrypted information to a file called *EncodedStudentRecords.txt.* The code should create the file or overwrite any existing file of the same name. Once the process is complete the display shows the following message:

"File encoded."

Example Output:

```
1.Would you like to see a list of all registered students?
2.Would you like to see a class list for a specific subject?
3.Would you like to see who your oldest student is?
4.Would you like to see who your youngest student is?
Enter the number for the selected task, or X to skip this:X

Would you like to encode a copy of the student records? Type Y for Yes or N or No :Y

File encoded.
```

IMPORTANT:
You should only overwrite the file when you start the list. For every record after the first record it should just append the information to the current file version.

Part 4

This requires you to add a function to your `EncodeDecodeClass`. The function should reverse the encoding done in Part 3 using a decode function called, *`DecodeStudentList()`* . The function call should receive a filename to work with, as a parameter. Decode the encoded files contents to a new file called, `DecodedStudentRecords.txt` Once the process is complete, display the following message:

"File decoded."

Example Output:

```
1.Would you like to see a list of all registered students?
2.Would you like to see a class list for a specific subject?
3.Would you like to see who your oldest student is?
4.Would you like to see who your youngest student is?
Enter the number for the selected task, or X to skip this:X

Would you like to encode a copy of the student records? Type Y for Yes or N or No :N

File decoded.
```

Make the calls and instantiation of it within *`StudentManagement.py`* as necessary.