PRÁCTICA #1

MANUAL TÉCNICO

Gestor de cursos

DAMIÁN IGNACIO PEÑA AFRE
202110568

CONTENIDO

ENCABEZADO

- Nombre del sistema: Gestor Cursos LFP
- Desarrollador: Damián Ignacio Peña Afre
- Lenguaje utilizado: Python
- Tipo de programa: Desktop
- Cuenta con interfaz gráfica

TECNICAS O PARADIGMAS

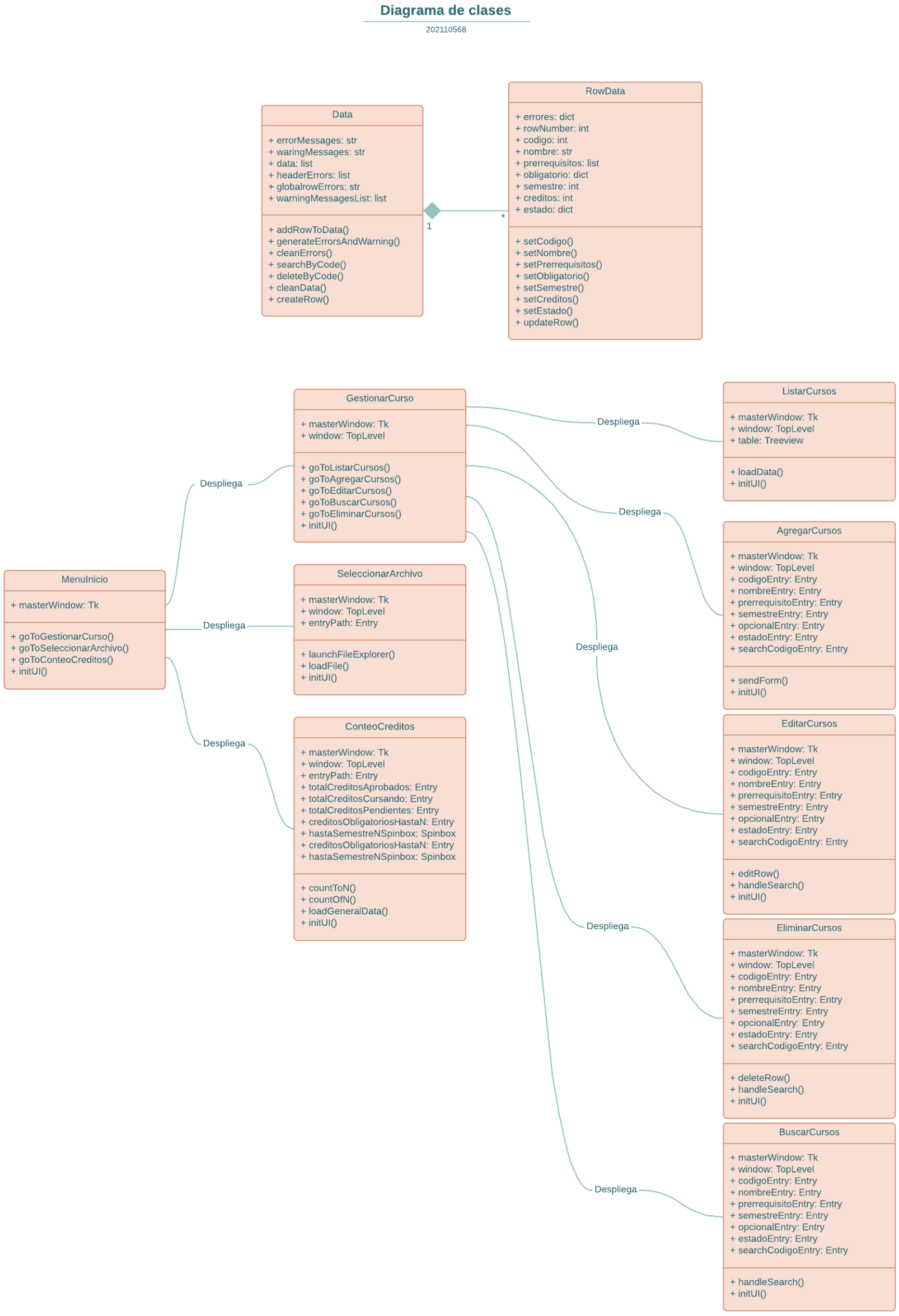
Se utilizo el paradigma de la programación orientada a objetos junto con la separación de los archivos en carpetas en función del tipo de tarea que realizan, de esta manera el proyecto esta segmentado de una forma modular.

Se separó la lógica del aspecto visual, de modo que obtiene una estructura mucho mas limpia de código.

CONVENCIONES DE NOMENCLATURA

Se optó por utilizar '*camelcase*' para nombrar los archivos y variables. Tanto el nombre de la clase como el archivo que lo contiene empiezan con letra mayúscula. Los métodos/funciones y archivos que contienen únicamente métodos/funciones empiezan por letra minúscula.

DIAGRAMA DE CLASES



METODOS PRINCIPALES

1. *loadFile*

- Descripción: Se comunica mediante el atributo *entryPath* de la clase *SeleccionarArchivo* para recuperar la ruta del archivo que se pretende leer. Luego realizar la validación de que se haya introducido una cadena no vacía se procede a abrir el archivo para almacenar todo su contenido dentro de una variable que será enviada al constructor de la clase *Data* la cual se encarga de darle formato para posteriormente crear un objeto *RowData*.

```
def loadFile(self):
        try:
            path = self.entryPath.get()

            if len(path) <= 0:
                messagebox.showwarning(title="Aviso",
                                       message="Debe introducir una ruta")
                return

        try:
            file = open(path, 'r', encoding="utf-8", errors='ignore')
            text = file.read()

            Data.cleanData()
            Data(text)
            file.close()

            if len(Data.warningMessagesList) > 0:
                messagebox.showwarning(title="Advertencia",
                                       message=Data.warningMessages)

            if len(Data.headerErrors) > 0 or len(Data.globalrowErrors) > 0:
                messagebox.showerror(title="Error",
                                     message=Data.errorMessages)
            else:
                messagebox.showinfo(title="Informacion cargada",
                                    message="Informacion cargada")

            Data.cleanErrors()
        except:
            messagebox.showerror(title="Error",
                                 message="Error en la carga del archivo")
```

2. createRow

- Descripción: A partir de una de una fila del archivo previamente formateada para que únicamente tenga los datos separados por comas y de un numero de correlativo se recuperan los datos necesarios. En caso de tener algun dato se procede a añadir un error a la lista de *headerErrors* que luego de recorrer todas las filas puede ser generada en texto para ser mostrada en un cuadro de dialogo. Este método retorna un objeto tipo *RowData* Si todo se realiza correctamente.

```
@staticmethod
    def createRow(rowWithCommas, rowNumber):
        row = rowWithCommas.split(',')
        try:
            codigo = row[0]
            nombre = row[1]
            prerrequisitos = row[2]
            obligatorio = row[3]
            semestre = row[4]
            creditos = row[5]
            estado = row[6]

            return RowData(rowNumber, codigo, nombre, prerrequisitos,
                           obligatorio, semestre, creditos, estado)

        except IndexError:
            Data.headerErrors.append({
                'rowNumber':
                rowNumber,
                'msg':
                'Hace falta un dato en la fila ' + str(rowNumber)
            })
            return None
```

3. searchByCode

- Descripción: Recibe como parámetro un código que es comparado con cada elemento que se añadió a la lista de datos. El objeto *RowData* que coincide con el código introducido es devuelto dentro de un diccionario, caso contrario regresa un diccionario con el mensaje de error.

```
@staticmethod
    def searchByCode(codigo):

        response = {'row': None, 'error': 'No se encontró el curso'}

        try:

            for row in Data.data:
                if row.codigo == int(codigo):
                    response['row'] = row
                    break
        except:
            pass

        return response
```

4. deleteByCode

- Descripción: Recibe como parámetro un código que es comparado con cada elemento que se añadió a la lista de datos. El objeto *RowData* que coincide con el código introducido es eliminado de la lista de datos. El método retorna *True* si es eliminado correctamente o *False* si no se encontró ninguna coincidencia.

```
    @staticmethod
    def deleteByCode(codigo):
        for row in Data.data:
            if row.codigo == int(codigo):
                Data.data.remove(row)
                return True

        return False
```

5. updateRow

- Descripción: Recibe como parámetro todos los atributos necesarios para la modificación y utiliza los *setters* de la clase *RowData* para realizar la modificación. Regresa un arreglo con los errores que se generaron por esta nueva modificación.

```
def updateRow(self, nombre, prerrequisitos, obligatorio, semestre,
                  creditos, estado):
        self.errores = {'rowNumber': None, 'list': []}

        self.setNombre(nombre)
        self.setPrerrequisitos(prerrequisitos)
        self.setObligatorio(obligatorio)
        self.setSemestre(semestre)
        self.setCreditos(creditos)
        self.setEstado(estado)

        return self.errores['list']
```

6. addRowToData

- Descripción: Recibe como parámetro un objeto *RowData*. Internamente el método decide en base a los errores que produjeron los *setters* del objeto la inserción de este. Asimismo, verifica que no haya sido introducido anteriormente y en dado caso lo esté, añade una nueva advertencia a la lista de advertencias.

```
    @staticmethod
    def addRowToData(newRow):
        if len(newRow.errores['list']) > 0:
            Data.globalrowErrors.append(newRow.errores)
        else:
            rowIndex = 0
            for rowInData in Data.data:
                if rowInData.codigo == newRow.codigo:
                    Data.warningMessagesList.append(
                        'Fila ' + str(rowIndex + 1) +
                        " sobreescrita por la fila " + str(newRow.rowNumber))
                    newRow.rowNumber = (rowIndex + 1)
                    Data.data[rowIndex] = newRow
                    return
                rowIndex += 1
            Data.data.append(newRow)
```

7. Nombre: countGeneralCredits

- Descripción: Recorre cada uno de los datos insertados y regresa en un diccionario la cantidad de créditos que tengan como estado: aprobados, cursando o pendientes (obligatorios).

```
def countGeneralCredits():

    aprobados = 0
    cursando = 0
    pendientes = 0
    for row in Data.data:

        estado = row.estado['number']
        opcionalidad = row.obligatorio['number']
        creditos = row.creditos

        if estado == -1 and opcionalidad == 1:
            pendientes += creditos
        elif estado == 1:
            cursando += creditos
        elif estado == 0:
            aprobados += creditos

    return {
        "aprobados": aprobados,
        "cursando": cursando,
        "pendientes": pendientes
    }
```

8. Nombre: countObligatoryCreditsToN

- Descripción: Recorre cada uno de los datos insertados y regresa la cantidad de créditos que tengan la opcionalidad como obligatoria hasta cierto semestre.

```
def countObligatoryCreditsToN(semesterNumber):
    creditos = 0
    try:
        for row in Data.data:
            if row.semestre <= int(semesterNumber):
                if row.obligatorio['number'] == 1:
                    creditos += row.creditos
    except:
        pass

    return creditos
```

9. Nombre: countAllCreditsOfN

- Descripción: Recorre cada uno de los datos insertados y regresa la totalidad de créditos de un semestre en particular.

```
def countAllCreditsOfN(semesterNumber):
    creditos = 0

    try:
        for row in Data.data:
            if row.semestre == int(semesterNumber):
                creditos += row.creditos
    except:
        pass

    return creditos
```

REQUERIMIENTOS

- Contar con un menú de opciones
- Cargar un archivo con extensión CSV o LFP
- Validar los campos:
    - Código: Identificador del curso
    - Nombre: Nombre del curso
    - Obligatorio: 0 (opcional) o 1 (obligatorio)
    - Semestre: Numero de semestre
    - Créditos: Créditos que aporta el curso
    - Estado: 0 (aprobado), 1 (cursando) o -1 (pendiente)
- Listar los cursos
- Mostrar los cursos
- Agregar curso
- Editar curso
- Eliminar curso
- Interacción amigable
- Conteo de créditos:
    - Total de créditos aprobados
    - Total de créditos cursando
    - Total de créditos (obligatorios)
    - Creditos obligatorios hasta cierto semestre
    - Creditos totales de semestre
- Utilización del lenguaje Python

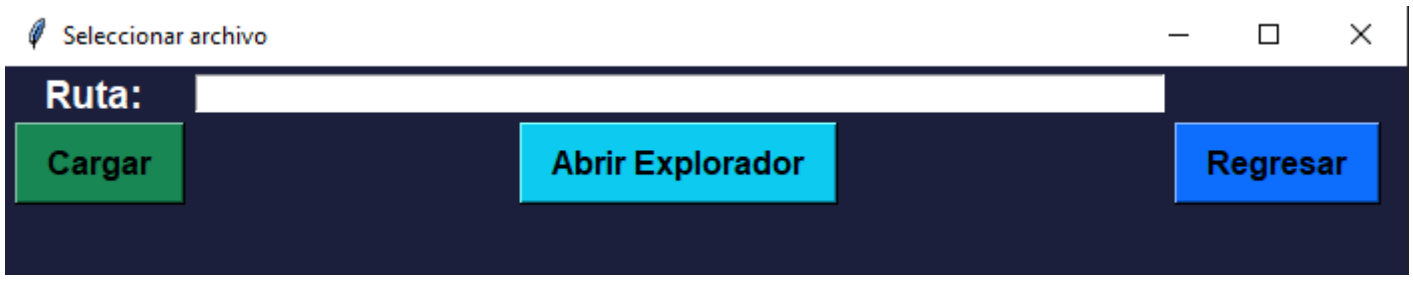
DESCRIPCIÓN

Para la realización del proyecto se utilizaron las siguientes herramientas:

1. Visual Studio Code: Editor de código ligero.
2. Git: Programa para llevar un control de las versiones del proyecto.
3. Github: Repositorio remoto para llevar un control de las versiones del proyecto en la nube.

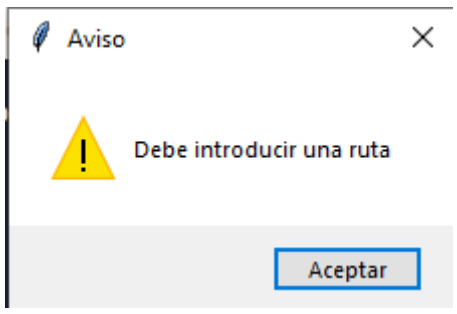
INTERFACES PRINCIPALES

1. *Seleccionar archivo*

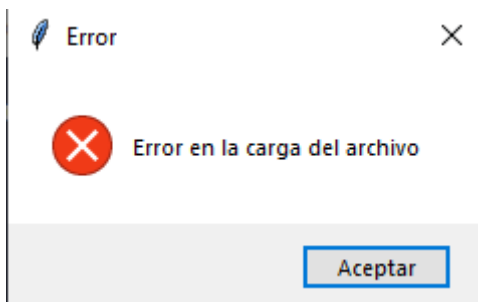


Validaciones:

1. Ruta vacía

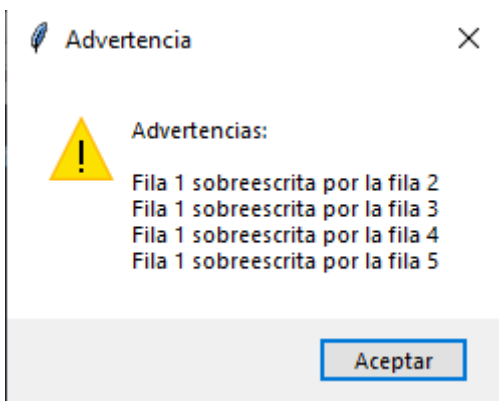


2. Ruta no valida

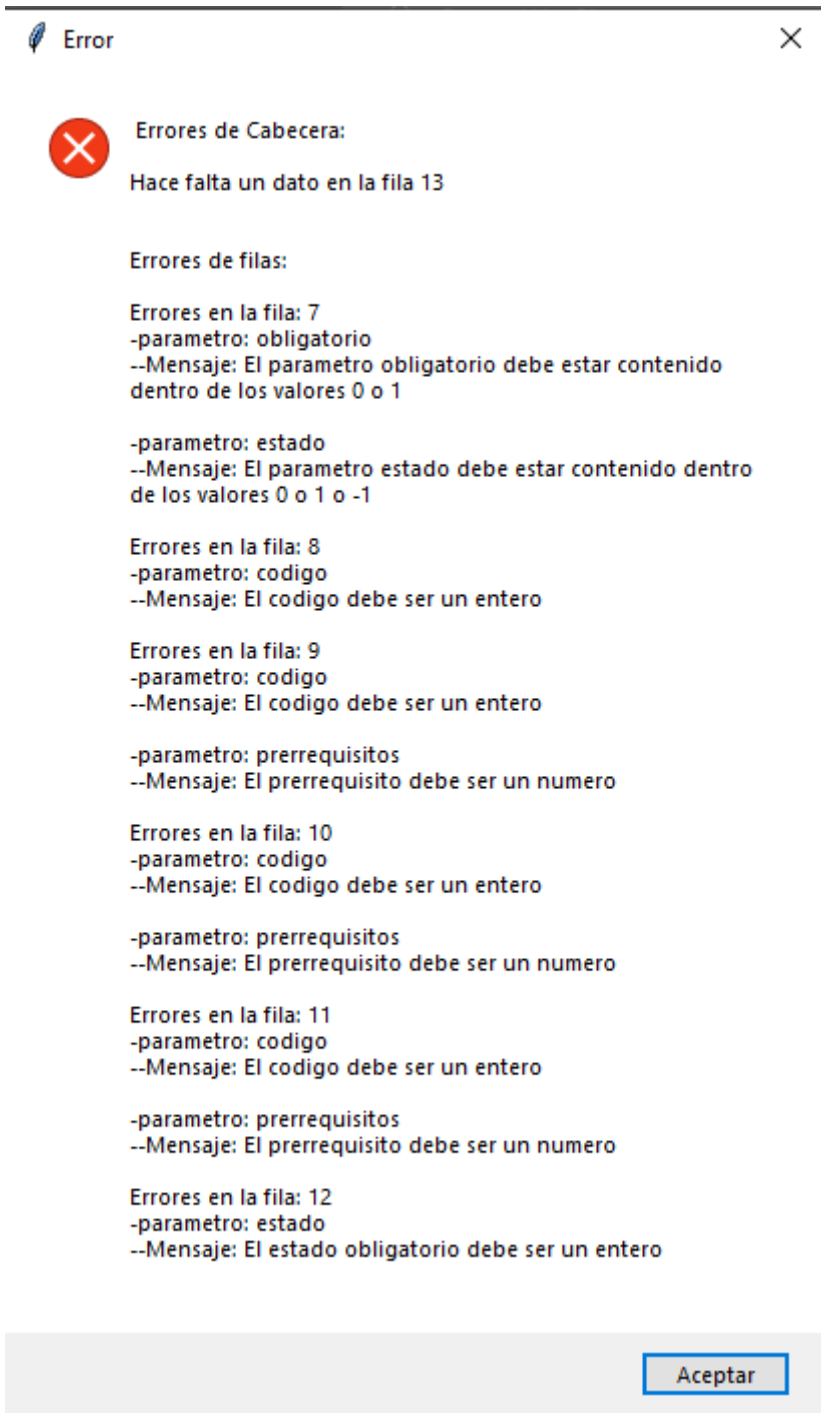


3. Archivo con extensión correcta, pero con errores

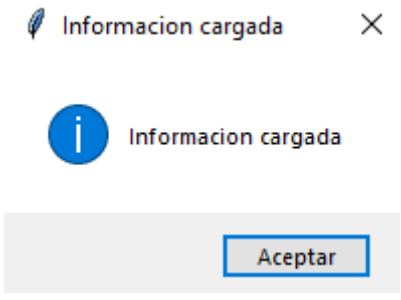
Sobreescritura de filas:



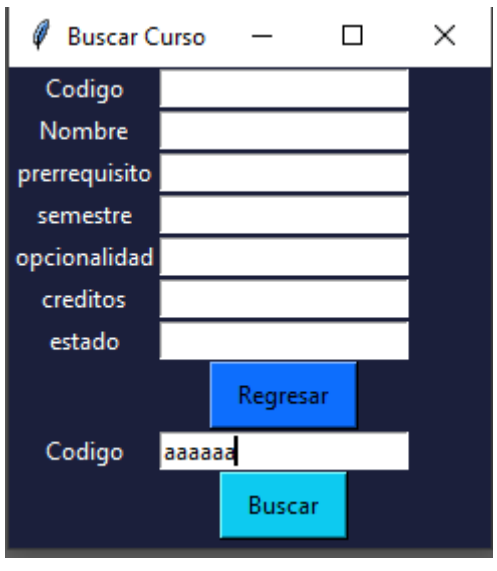
Errores generales:



4. Archivo correcto

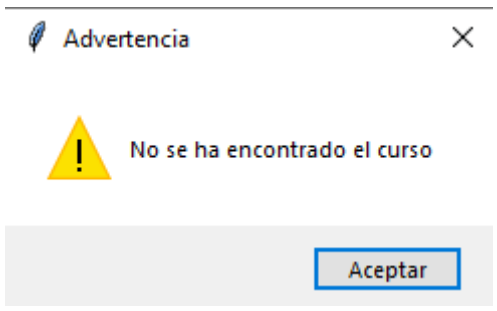


2. *Buscar curso*

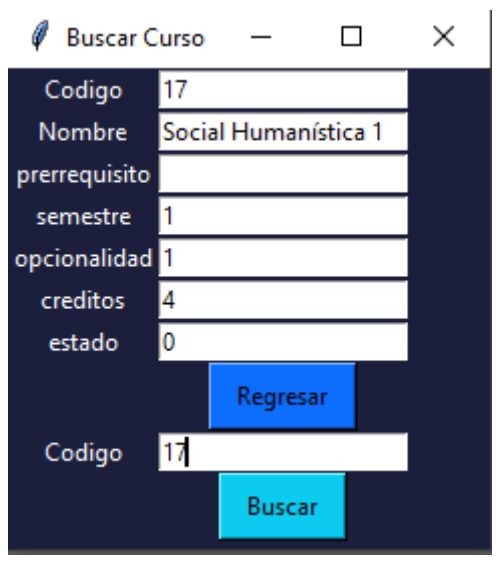


Validaciones

1. Código incorrecto

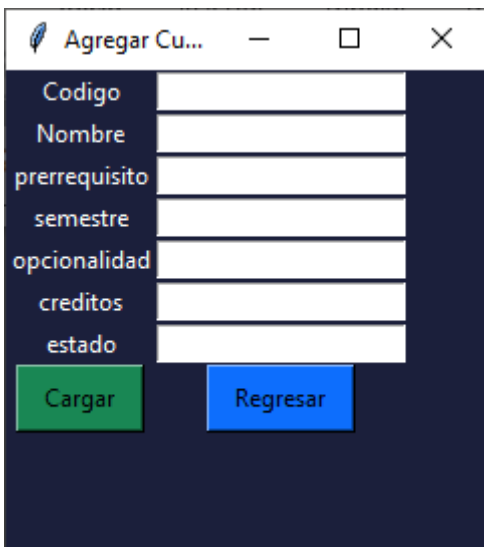


2. Código correcto



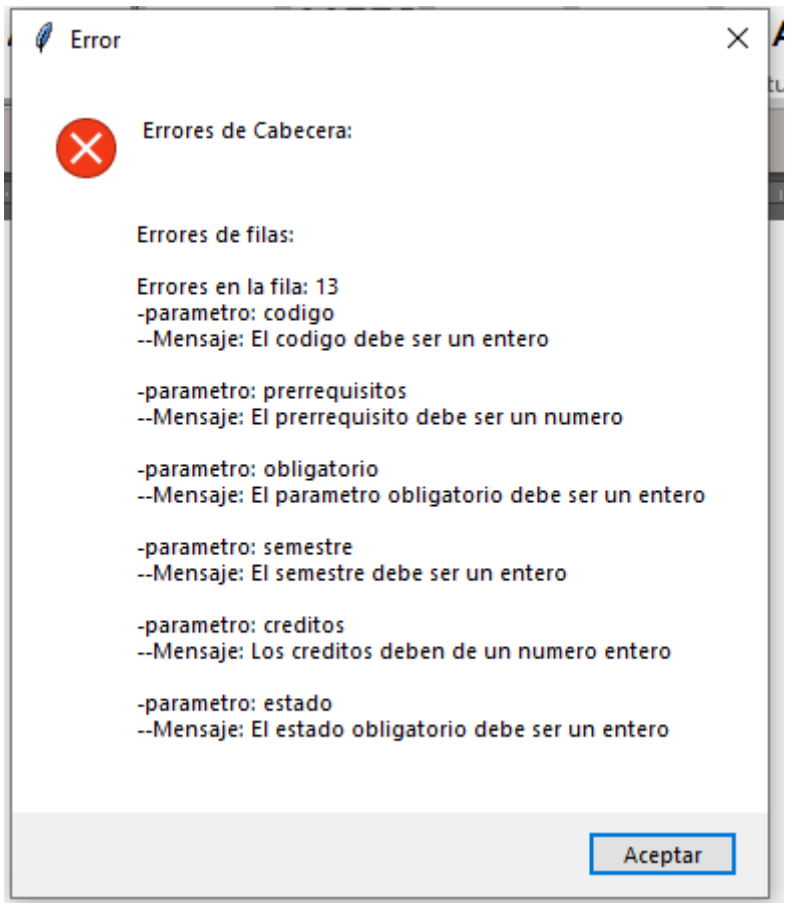
*Esta funcionalidad la tienen los formularios de: editar curso y eliminar curso**

3. Agregar Curso

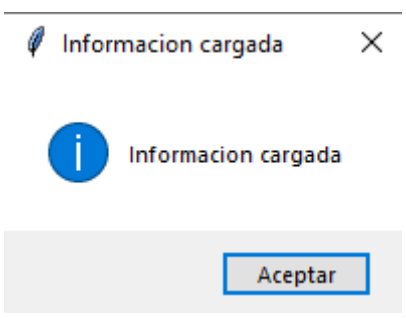


Validaciones

1. Tipos de datos incorrectos



2. Datos correctos



*Esta funcionalidad la tiene el formulario de editar, con la excepción de que altera el registro previo**

PLANIFICACIÓN

Semana #1 (1-7 agosto)

Día	Tarea	Tiempo empleado (Horas)
Lunes 1	Diseño de la interfaz del menú de inicio y de seleccionar archivo	1
Martes 2	Logica del manejo y carga de datos	1
Miercoles 3		
Jueves 4	Corrección de bugs y diseño de el resto de interfaces	1
Viernes 5	Listado de datos, agregar cursos	1
Sabado 6		
Domingo 7		

Semana #2 (8-14 agosto)

Día	Tarea	Tiempo empleado (Horas)
Lunes 8	Edición y eliminación de los cursos. Conteo de cursos	2
Martes 9		
Miercoles 10		
Jueves 11		
Viernes 12		
Sabado 13		
Domingo 14		

Semana #3 (15-21 agosto)

Día	Tarea	Tiempo empleado (Horas)
Lunes 8		
Martes 9		
Miercoles 10		
Jueves 11		
Viernes 12	Últimas modificaciones	1
Sabado 13	Documentación	3
Domingo 14	Entrega	