Manual Técnico

Datos de desarrollador

Lenguajes Formales y de Programación

Alvaro Norberto García Meza

Carné: 202109567

Detalles de desarrollo:

Se utilizo el lenguaje de programación Python y su herramienta integrada de desarrollo de interfaces gráficas, Tkinter, donde se realizo un CRUD con muchas más funcionalidades sobre el manejo de la lectura de archivos “.cvs”.

Descripción general:

Este programa carga y lee archivos de tipo “.cvs” donde posteriormente cada fila se convierte en una clase “Curso” la cual se va almacenando en un arreglo global durante la ejecución, donde en cada sección podremos, visualizar cursos, editarlos, agregarlos y eliminarlos. Implementando paradigmas de programación para la reutilización de código.

Paradigma utilizado:

Se implementaron dos paradigmas de programación el primero fue programación orientada a objetos donde se implemento dos clases una que hederá de la clase “Tk” para poder crear interfaces gráficas, dentro de esta clase se encuentra todas las herramientas que esta Liberia utiliza separada por módulos donde cada modulo representa una ventana anidada. Luego se utilizo una clase para el modelado de un curso donde este sea reutilizable durante la ejecución.

Por último, se trabajó con programación funcional para dividir las funciones de las ventanas fuera de su instancia como ventana misma.

Diccionario de librerías:

- **Tkinter**
  Es una librería integrada de Python la cual sirve para el desarrollo de interfaces gráficas. Proporciona un conjunto de herramientas e independientes para administrar ventanas.
  Widgets utilizados:
  - Clase tkinter como “tk.Tk” para inicializar un objeto como una aplicación.
  - Label: Para insertar texto estático en las ventanas.
  - Button: Botones para darle funcionalidad a nuestra aplicación al momento de darle clic.
  - Toplevel: Es un widget para crear ventanas hijas de la ventana principal.
  - Entry: Es un widget para ingresar valores de tipo texto.
  - Treeview: Para crear listas o en este caso simular filas y columnas de una tabla.
  - StringVar: Es una variable especial de tkinter para almacenar valores de tipo String.
  - Combobox: Es una combinación de una caja de texto y una lista de diferentes opciones para escoger una de ellas a la vez.
  - MessageBox: Es una función que lanza una altera en forma de una ventana externa para dar avisos al ocurrir algo.
  - SpinBox: Es un widget que permite seleccionar entre un rango de números.

Diccionario de clases:

- Clase “Subject” la cual hace referencia a los cursos leídos del archivo CVS, cada instancia de esta clase representa un curso del pensum de la carrera, con sus diferentes atributos encapsulados.

```python
class Subject():
    #Constructor
    def __init__(self,codigo,nombre,preRequisito,obligatorio,semestre,creditos,estado):
        self.__codigo = codigo
        self.__nombre = nombre
        self.__preRequisito = preRequisito
        self.__obligatorio = obligatorio
        self.__semestre = semestre
        self.__creditos = creditos
        self.__estado = estado
    #Getters and Setters
    #Code
    @property
    def codigo(self):
        return self.__codigo

    @codigo.setter
    def codigo(self,codigo):
        self.__codigo = codigo

    #name
    @property
    def nombre(self):
        return self.__nombre

    @nombre.setter
    def nombre(self,nombre):
        self.__nombre = nombre

    #Prerequisito
    @property
    def preRequisito(self):
        return self.__preRequisito

    @preRequisito.setter
    def preRequisito(self,pre):
        self.__preRequisito = pre

    #Obligatorio
    @property
    def obligatorio(self):
        if self.__obligatorio == '1':
            return 'Obligatorio'
        elif self.__obligatorio == '0':
            return 'Opcional'
```

- Clase “MainPage” es la clase principal donde esta creado la interfaz gráfica heredando de la clase tk.Tk, está clase maneja todas las ventanas y widgets de la librería Tkinter y ciertas funcionalidades.

```
class MainPage(tk.Tk):

    def __init__(self):
        super().__init__()
        #Title
        self.title('PRACTICA 1')
        #Dimension
        self.geometry('600x400+700+200')
        self['background']='#dbdbdb'
        #Grid
        self.columnconfigure(0,weight=1)

        #Variables
        self._var = tk.StringVar(value='Ruta del archivo...')
        #Containt the file
        self.file = None
        #To verified the actual file
        self.open_file = False
        #list of subject objects
        self.subjects = []
        #Components
        self._create_components()
```

Diccionario de variables:

- Las variables globales utilizadas para la interfaz gráfica solo fueron 4: “self._var” para manejar la dirección del archivo abierto durante la ejecución de nuestro programa, “self.file” y “self.open_file” trabajan de manera sincrónica para validar que hay un archivo abierto y que no sea vuelva a leer el mismo. Por ultimo tenemos “self.subjects” el cual es el arreglo de objetos “Subject” el cual almacenará todos los cursos leídos y agregados durante la ejecución de nuestro programa.

```
#Variables
self._var = tk.StringVar(value='Ruta del archivo...')
#Containt the file
self.file = None
#To verified the actual file
self.open_file = False
#list of subject objects
self.subjects = []
```

Diccionario de métodos y funciones:

- El método “select_file” tiene la funcionalidad de abrir nuestro explorador de archivos escoger el archivo a leer y asignárselo a una de nuestras variables globales para mantener consistencia de la ruta del archivo abierto. Posteriormente se lee el archivo, se usa el método “Split” para crear una lista de por cada salto de línea en nuestro archivo, luego mediante un for se borran las líneas vacías, luego se vuelve a usar el método “Split” para crear una lista de listas de los valores separados por comas. Luego se procede a borrar los duplicados, donde en una lista externa se copian todos los códigos de la lista, se filtran para los cursos repetidos y así mismo se vuelve a crear una lista auxiliar que guarda los datos no repetidos. Por último, se recorre la lista filtrada “new” y por cada lista se crea un objeto “Subject” y se almacena en la lista global de objetos y se procede a mandar un mensaje aprobación.

```python
def select_file(self):
    # We open the file
    self.open_file = askopenfile(mode='r+')
    # Chekend if there is a file
    if not self.open_file:
        return
    # If the same file has already been opened only return
    if self._var.get() == self.open_file.name:
        return
    # Open the file
    with open(self.open_file.name, 'r+',encoding='utf-8') as file:
        # We read the file
        files = file.read()
        print(files)

        data = []
        nums = []
        new_nums = []
        new = []
        # Split each course, then asign to a void list -> data
        response = files.split('\n')
        #Delete void lines
        for i in response:
            if i == '':
                response.remove(i)
        #Split each value of each array
        for i in response:
            data.append(i.split(','))
        # First fill a list with only the code of the courses
        for i in data:
            nums.append(i[0])
        # If the code is duplicated, only use the first of them
        for i in range(len(nums)):
            if nums[i] not in new_nums:
                new_nums.append(nums[i])
                new.append(data[i])
        print(new)
        # Iterate data to convert each value of each list an object Subject, then asign to self.subjects
        for i in new:
            # print(f' {i[0],i[1],str(i[2]),i[3],i[4],i[5],i[6]} ')
            obj = Subject(i[0], i[1], i[2], i[3],i[4],i[5],i[6])
            self.subjects.append(obj)

        # Change the rute
        self._var.set(file.name)
        messagebox.showinfo(
            'Selección de archivo', 'Se ha cargado y leído correctamente el archivo.')
```

- El método “add_an_existing_subject” verifica que, al momento de agregar un curso, si la lista global no esta vacía busca que el código del curso no se haya agregado y si es así entonces eliminará el curso anteriormente agregado y volverá a enlistar, pero como un curso nuevo, si el curso agregado no se encuentra ya en la lista, simplemente lo agrega como un curso más. Pero si la lista se encuentra vacía simplemente agregará el curso como primero.

```
def add_an_existing_subject(self, code, name, optional, semester, state, pre, credits):
    obligatorio = ''
    estado = ''
    # Transform the values
    if optional.get() == 'Opcional':
        obligatorio = '0'
    elif optional.get() == 'Obligatorio':
        obligatorio = '1'

    if state.get() == 'Aprobado':
        estado = '0'
    elif state.get() == 'Cursando':
        estado = '1'
    elif state.get() == 'Pendiente':
        estado = '-1'

    if len(self.subjects) > 0:
        for i in self.subjects:
            if i.codigo == code.get():
                messagebox.showerror(
                    'Agregar Curso',
                    f'Curso con código {i.codigo} ya existe. Se Eliminará y Reemplazará '
                )
                self.subjects.remove(i)
                # Add Course
                obj = Subject(code.get(), name.get(), pre.get(), obligatorio, semester.get(), credits.get(), estado)
                self.subjects.append(obj)
                # Restart
                code.set('')
                name.set('')
                pre.set('')
                optional.set('')
                semester.set('')
                credits.set('')
                state.set('')
                obligatorio = ''
                estado = ''
                break
            else:
                # If there not an existing subject, we validate and add to the list
                # Add Course
                obj = Subject(code.get(), name.get(), pre.get(), obligatorio, semester.get(), credits.get(), estado)
                self.subjects.append(obj)
                # Restart
                code.set('')
                name.set('')
                pre.set('')
                optional.set('')
                semester.set('')
                credits.set('')
                state.set('')
                obligatorio = ''
                estado = ''
                messagebox.showinfo(
                    'Agregar Curso',
                    f'Se ha agregado el curso con código{obj.codigo} correctamente!.'
                )
                break
    else:
        # If there not an existing subject, we validate and add to the list
        # Add Course
        obj = Subject(code.get(), name.get(), pre.get(),obligatorio, semester.get(), credits.get(), estado)
        self.subjects.append(obj)
        # Restart
        code.set('')
        name.set('')
        pre.set('')
        optional.set('')
        semester.set('')
        credits.set('')
        state.set('')
        obligatorio = ''
        estado = ''
        messagebox.showinfo(
            'Agregar Curso',
            f'Se ha agregado el curso con código{obj.codigo} correctamente!.'
        )
```

- El método “select_and_show_course” recorre la lista de objetos y filtra al objeto que se seleccionó en nuestro “combobox” y cambia las variables por los datos de dicho curso.

```
def select_and_show_course(self, code, name, optional, semester, state, pre, credits,comboget):
    for i in self.subjects:
        if i.nombre == comboget:
            code.set(f'Código: {i.codigo}')
            name.set(f'Nombre: {i.nombre}')
            pre.set(f'Pre requisito: {i.preRequisito}')
            optional.set(f'Opcionalidad: {i.obligatorio}')
            semester.set(f'Semestre: {i.semestre}')
            credits.set(f'Créditos: {i.creditos}')
            state.set(f'Estado: {i.estado}')
```

- El método “edit_subject” funciona de la misma manera que agregar curso, donde se filtra el curso seleccionado y se edita por los parámetros mandados en la función.

```
def edit_subject(self, code, name, optional, semester, state, pre, credits):
    # Var
    obligatorio = ''
    estado = ''

    # Transform the values
    if optional.get() == 'Opcional':
        obligatorio = '0'
    elif optional.get() == 'Obligatorio':
        obligatorio = '1'

    if state.get() == 'Aprobado':
        estado = '0'
    elif state.get() == 'Cursando':
        estado = '1'
    elif state.get() == 'Pendiente':
        estado = '-1'
    # Iterate through the subjects, find the course and edit their values
    for i in self.subjects:
        # If the code is equal to the code.get
        if i.codigo == code.get():
            # Edit
            i.codigo = code.get()
            i.nombre = name.get()
            i.preRequisito = pre.get()
            i.obligatorio = obligatorio
            i.semestre = semester.get()
            i.creditos = credits.get()
            i.estado = estado
            messagebox.showinfo('Curso editado',f'Se ha editado correctamente el curso {i.nombre}!')
            # Restar the values
            code.set('')
            name.set('')
            pre.set('')
            optional.set('')
            semester.set('')
            credits.set('')
            state.set('')
            obligatorio = ''
            estado = ''
            break
```

- El método “delete” filtra el curso seleccionado de la lista y lo remueve mediante su valor con el método “remove”.

```
def delete():
    select = combo.get()
    for i in self.subjects:
        if i.nombre == select:
            self.subjects.remove(i)
            messagebox.showwarning('Eliminar Curso',f'Se ha eliminado el curso {i.nombre} correctamente!')
```

- El método “approved_fun” cuenta todos los créditos que se encuentran dentro de nuestra lista global de objetos en ejecución y los clasifica por créditos aprobados, cursando y pendientes (donde estén en obligatorio). Se almacenan los valores en variables locales para luego ser mostradas en pantalla.

```
def approved_fun(self,approved_sub,cursing_sub,pending_sub):
    a = 0
    c = 0
    p = 0
    for i in self.subjects:
        if i.estado == 'Aprobado':
            a += int(i.creditos)
        elif i.estado == 'Cursando':
            c  += int(i.creditos)
        elif i.estado == 'Pendiente' and i.obligatorio == 'Obligatorio':
            p  += int(i.creditos)

    approved_sub.set(a)
    cursing_sub.set(c)
    pending_sub.set(p)
```

- El método “selection_changed” obtiene el semestre N y calcula los créditos de índole “obligatorio” desde el semestre 1 al semestre N y los suma en una variable local de la función.

```
def selection_changed(event):
    total_credits = 0
    n_semester = combo.get()
    #Calculated the credits of the semester 1 to N, only the required courses
    for i in self.subjects:
        if int(i.semestre) <= int(n_semester) and i.obligatorio == 'Obligatorio':
            total_credits += int(i.creditos)

    required_credtis.set(f'Créditos Obligatorios hasta el semestre {n_semester}: {total_credits} créditos ')
```

- El método “count_credits_per_semester” cuenta los créditos de los cursos aprobados, cursando y pendientes del semestre específicamente seleccionado.

```
def count_credits_per_semester():
    #semester
    semester = spin.get()
    if int(semester) > 10 or int(semester) < 1:
        messagebox.showerror('Error!!',f'El smestre {semester} no es válido')
    else:
        c_a = 0
        c_c = 0
        c_p = 0
        #Filter by semester
        for i in self.subjects:
            if int(i.semestre) == int(semester):
                if i.estado == 'Aprobado':
                    c_a += int(i.creditos)
                elif i.estado == 'Cursando':
                    c_c += int(i.creditos)
                elif i.estado == 'Pendiente':
                    c_p += int(i.creditos)
        print(semester)
        messagebox.showinfo(f'Creditos del semestre {semester}', f'Creditos de cursos aprobados {c_a} \n Créditos de cursos asignados {c_c} \n Créditos de cursos pendientes {c_p} ')
```