UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS Y SISTEMAS
LENGUAJES FORMALES Y DE PROGRAMACIÓN
PRIMER SEMESTRE 2,022.

Manual Tecnico

Autor: Brayan Estiben Micá Pérez

201907343

Introducción

El presente manual se presentan las especificaciones técnicas del programa de carga de archivos con extensión. flp, se muestra cada función por separado para poder hacer modificaciones al código posteriormente.

Objetivos

- Otorgar soporte técnico a los desarrolladores para poder modificar la aplicación.

Requerimientos

- Equipo: Dual Core o superior
- Sistema Operativo: Windows 7 o superior
- Memoria RAM: 1 Gb mínimo o superior
- Resolución gráfica: mínimo 800*600 o superior
- Tener instalado Python 3 con todas sus librerías
- Tener instalado el gestor de paquetes pip para descargar extensiones

Opciones de sistema

El presente manual está organizado de acuerdo a la secuencia de ingreso a las pantallas del sistema de la siguiente manera:

1. Opciones de sistema
2. Cargar archivo
3. Mostrar datos del archivo
4. Eliminar un registro
5. Agregar un registro
6. Modificar un registro
7. Ver el conteo de créditos

1. Opciones de sistema

   Se detalla un menú utilizando tkinter con botones y labels asociados cada unos de ellos a una función en específico. Todos estos objetos se crean en una clase llamada Myapp.

```python
# Create Object
class Myapp():
    def __init__(self):
        self.abrir = cargaArchivo()
        self.root = Tk()
        # Set title
        self.root.title("Menu principal")
        self.root.resizable(False, False)
        # Set Geometry
        self.root.geometry("500x300")
```

```python
LabelCurso = Label(self.root,text="Nombre del Curso: Lab. Lenguajes Formales y de Prog
LabelCurso.grid(row=0, column=0,sticky="w",padx=10,pady=10)
LabelEstudiante = Label(self.root,text="Nombre del Estudiante: Brayan Estiben Micá Pér
LabelEstudiante.grid(row=1, column=0,sticky="w",padx=10,pady=10)
LabelCarne = Label(self.root,text="Carné del Estudiante: 201907343")
LabelCarne.grid(row=2, column=0,sticky="w",padx=10,pady=10)

# Botones de menu principal
btnCargarArchivo = Button(self.root, text="Cargar archivo", command=self.abrir.abrir)
btnCargarArchivo.grid(row=3, column=1,sticky="w",padx=5,pady=5)
btnGestionar = Button(self.root, text="Gestionar Cursos", command=self.Gestionar)
btnGestionar.grid(row=4,column=1,sticky="w",padx=5,pady=5)
btnConteoCreditos = Button(self.root, text="Conteo De Creditos")
btnConteoCreditos.grid(row=5, column=1,sticky="w",padx=5,pady=5)
btnSalir = Button(self.root, text="Salir", command=self.root.destroy)
btnSalir.grid(row=6,column=1,sticky="w",padx=5,pady=5)
```

2. Cargar archivo

   La opción de cargar archivo verifica la extensión del archivo asi como la selección del mismo además se tiene una función que recupera dicha respuesta si el archivo es seleccionado o no.

```
class cargaArchivo():
    def abrir(self):
        Tk().withdraw()
        self.archivo = filedialog.askopenfile(
            title="Selecionar un archivo",
            initialdir="./",
            filetypes=(
                ("archivos TXT",".LFP" or ".lfp"),
                ("todos los archivos",".LFP" or ".lfp")
                # en esta parte colocar el tipo de archivo
                # .html
                # .docs
            )
        )
```

Función de verificación de selección del archivo, además al finalizar la carga del archivo se convertirá el texto recibido a UTF-8

```
try:
    if self.archivo is None:
        error = Tk()
        error.title("Venta de Error de Carga")
        error.geometry("200x200")
        error.configure(bg='red')
        lb_error = Label(error, text="Hubo un error al cargar el archivo",foregrour
        lb_error.grid(row=0, column=0,sticky="w",padx=10,pady=10)
        btn_error = Button(error, text="Aceptar", command=error.destroy)
        btn_error.grid(row=1, column=0,sticky="w",padx=10,pady=10)
    else:
        with open(self.archivo.name,"r",encoding="utf8") as file:
            self.texto = file.read()
            file.close()
```

3. Mostrar datos del archivo

Para la muestra del contenido del archivo cargado en memoria se leerá a través de una función leer para verificar el contenido y asi poder convertirlo a una lista cada curso con su código y demás categorías de guardaran dentro de un árbol y posteriormente cada árbol de tendrá en una única lista para su posterior manipulación.

```
def leerTexto(self):
    try:
        self.texto =  self.texto.split("\n")
        contador=0
        self.lista2=[]
        if len(self.texto)>0:
            for c in self.texto:
                diccionario = {"Codigo":None, "Nombre":"", "Prerrequisitos":None, "Oblig
                listaC = c.split(",")
                #print("lista ",contador," ",listaC)
                for cr in listaC:
                    #print(cr)
                    if(contador == 0):
                        diccionario["Codigo"]=cr
                        contador= contador+1
                    elif(contador == 1):
                        diccionario["Nombre"]=cr
                        contador= contador+1
                    elif(contador == 2):
                        diccionario["Prerrequisitos"]= cr.split(";")
```

Al finalizar la lectura del archivo se tendrá un capturador de errores si en dado caso se necesita.

```
                    elif(contador == 5):
                        diccionario["Creditos"]=int(cr)
                        contador= contador+1
                    elif(contador == 6):
                        diccionario["Estado"]=int(cr)
                        contador =0
                self.lista2.append(diccionario)
                #print(diccionario)
        return self.lista2
    except:
        error = Tk()
        error.title("Venta de Error de Lectura de archivo")
        error.geometry("250x150")
        error.configure(bg='red')
        lb_error = Label(error, text="No tienes cargado a memoria el archivo",foreground
        lb_error.grid(row=0, column=0,sticky="w",padx=10,pady=10)
        btn_error = Button(error, text="Aceptar", command=error.destroy)
        btn_error.grid(row=1, column=0,sticky="w",padx=10,pady=10)
```

4. Eliminar un registro

Para eliminar un registro se utilizó una llama al getDatos para buscar los registros en la lista y posteriormente se eliminó a través de la creación de una nueva lista cuando se encontró dicho registro, al encontrar la posición del registro a eliminar se utilizó la instrucción pop para eliminar el registro y posteriormente se hizo una actualización a el registro setDatos().

```
def borrarCurso(self):
    temporal = False
    self.eli = []
    self.eli.clear()
    self.eli=self.abrir.getDatos()
# iterando toda la lista para encotrar el valor
    contador=0
    for i in self.eli:
        # Verificando en que posicion se encuentra el curso a eliminar
        if self.entrada.get() == i["Codigo"]:
            # eliminar elemento encontrado
            self.eli.pop(contador)
            self.abrir.setDatos(self.eli)
            # print("El registro ",i, " ha sido eliminado")
            # limpiando cada de texto
            tkinter.messagebox.showinfo(title="Curso eliminado", message=("El curso ha
            self.entrada.delete(0, END)
            temporal=True
            break
        contador=contador+1
```

Si en dado caso no se encontrara con el código del registro a eliminar se muestra un mesaje de error utilizando meesagebox

```
    contador=contador+1
if temporal == False:
    tkinter.messagebox.showinfo(title="Curso eliminado", message=("El curso que escribio no exi
```

Para esta práctica se utilizó la programación orientada a objetos como base para el desarrollo de la aplicación, además de la combinación de la librería Tkinter para la visualización de la parte grafica