Tecnológico José Mario molina Pasquel y Henríquez UA Zapotlanejo

Manual de Programador

Carrera: Ingeniería en Informática

Materia: Programación orientada a objetos

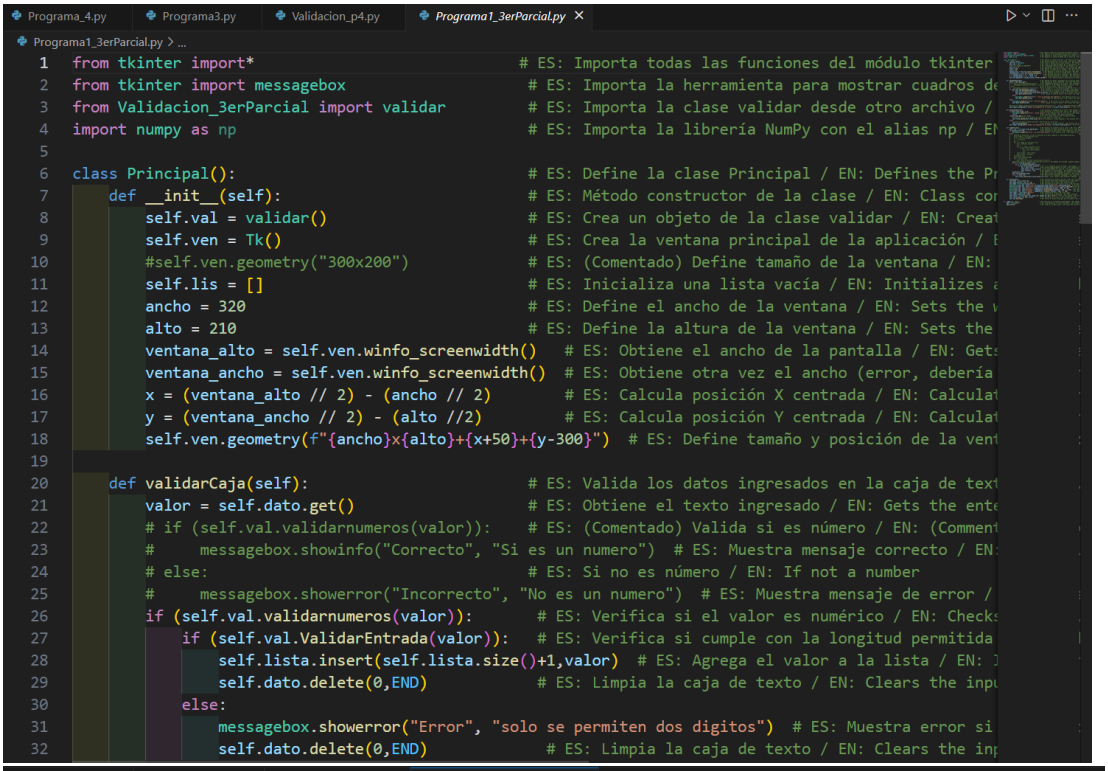
Docente: Osvaldo Rene Rojo Roa

Alumno: Carlos Israel Morales Chavez

INDICE

Contents

Programa 1



```python
from tkinter import*                                # ES: Importa todas las funciones del módulo tkinter
from tkinter import messagebox                      # ES: Importa la herramienta para mostrar cuadros de
from Validacion_3erParcial import validar           # ES: Importa la clase validar desde otro archivo /
import numpy as np                                  # ES: Importa la librería NumPy con el alias np / EN

class Principal():                                  # ES: Define la clase Principal / EN: Defines the Pr
    def __init__(self):                             # ES: Método constructor de la clase / EN: Class cor
        self.val = validar()                        # ES: Crea un objeto de la clase validar / EN: Creat
        self.ven = Tk()                             # ES: Crea la ventana principal de la aplicación / E
        #self.ven.geometry("300x200")               # ES: (Comentado) Define tamaño de la ventana / EN:
        self.lis = []                               # ES: Inicializa una lista vacía / EN: Initializes a
        ancho = 320                                 # ES: Define el ancho de la ventana / EN: Sets the v
        alto = 210                                  # ES: Define la altura de la ventana / EN: Sets the
        ventana_alto = self.ven.winfo_screenwidth()   # ES: Obtiene el ancho de la pantalla / EN: Gets
        ventana_ancho = self.ven.winfo_screenwidth()  # ES: Obtiene otra vez el ancho (error, debería
        x = (ventana_alto // 2) - (ancho // 2)        # ES: Calcula posición X centrada / EN: Calculat
        y = (ventana_ancho // 2) - (alto //2)         # ES: Calcula posición Y centrada / EN: Calculat
        self.ven.geometry(f"{ancho}x{alto}+{x+50}+{y-300}")  # ES: Define tamaño y posición de la vent

    def validarCaja(self):                          # ES: Valida los datos ingresados en la caja de text
        valor = self.dato.get()                     # ES: Obtiene el texto ingresado / EN: Gets the ente
        # if (self.val.validarnumeros(valor)):      # ES: (Comentado) Valida si es número / EN: (Comment
        #     messagebox.showinfo("Correcto", "Si es un numero")  # ES: Muestra mensaje correcto / EN:
        # else:                                     # ES: Si no es número / EN: If not a number
        #     messagebox.showerror("Incorrecto", "No es un numero")  # ES: Muestra mensaje de error /
        if (self.val.validarnumeros(valor)):        # ES: Verifica si el valor es numérico / EN: Checks
            if (self.val.ValidarEntrada(valor)):    # ES: Verifica si cumple con la longitud permitida
                self.lista.insert(self.lista.size()+1,valor)  # ES: Agrega el valor a la lista / EN:
                self.dato.delete(0,END)             # ES: Limpia la caja de texto / EN: Clears the inpu
            else:
                messagebox.showerror("Error", "solo se permiten dos digitos")  # ES: Muestra error si
                self.dato.delete(0,END)             # ES: Limpia la caja de texto / EN: Clears the inp
```

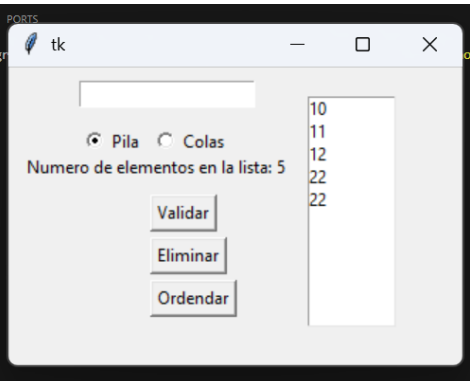
```python
class Principal():                                  # ES: Define la clase Principal / EN: Defines the Pr
    def validarCaja(self):                          # ES: Valida los datos ingresados en la caja de text
        else:
            messagebox.showerror("Error", "No son numeros")   # ES: Error si no es número / EN: Error
            self.dato.delete(0,END)                 # ES: Limpia la caja de texto / EN: Clears the inpu

        #print(f'la cadena tiene{str(self.val.ValidarEntrada(valor))}')   # ES: (Comentado) Muestra va
        self.label.config(text=f'Numero de elementos en la lista: {str(self.lista.size())}')  # ES: Ac

    def eliminardato(self):                         # ES: Elimina un elemento de la lista / EN: Deletes
        if self.lista.size() <= 0:                  # ES: Verifica si la lista está vacía / EN: Checks i
            messagebox.showerror("Error", "Lista vacia")  # ES: Muestra error si está vacía / EN: Shov
            return                                  # ES: Sale del método / EN: Exits the method
        if(self.modo.get()) == 'Pila':              # ES: Si el modo es pila / EN: If mode is stack
            #ultimo que entra es el primero en salir / EN: Last in, first out
            self.lista.delete(self.lista.size()-1) # ES: Elimina el último elemento / EN: Deletes the
        else:
            #primero que entra primero que sale / EN: First in, first out
            self.lista.delete(0)                    # ES: Elimina el primer elemento / EN: Deletes the
        self.label.config(text=f'Numero de elementos en la lista: {str(self.lista.size())}')  # ES: Ac

    def ordenar(self):                              # ES: Ordena los elementos de la lista / EN: Sorts
        self. lis = list(self.lista.get(0,END))     # ES: Obtiene los datos actuales de la lista / EN: F
        if len(self.lis) <=0:                       # ES: Verifica si está vacía / EN: Checks if list is
            messagebox.showerror('Error','La lista esta vacia')  # ES: Muestra error / EN: Shows error
        else:
        #     #metodo de seleccion, se usa un auxiliar en 0 para comparar e intercambia posicion
        #     self.arreglo = np.array(self.lis)
        #     # for i in self.arreglo:
        #     #     print(i)}
        #     p = 0
        #     for i in range (0, len(self.lis)):
```

```python
class Principal():                              # ES: Define la clase Principal / EN: Defines the Pr
    def ordenar(self):                          # ES: Ordena los elementos de la lista / EN: Sorts t
        #         aux = int(self.lis[i])
        #         p = i
        #         for x in range(i,len(self.lis)):
        #             if aux < int (self.lis[x]):
        #                 aux = int (self.lis[x])
        #                 p = x
        #         self.lis[p] = self.lis[i]
        #         self.lis[i] = str(aux)
        #     print(self.lis)
        #     self.lista.delete(0,END)
        #     for i in self.lis:
        #         self.lista.insert(self.lista.size()+1,str(i))
        #metodo de burbuja, compara una posicion con otra / EN: Bubble sort method, compares adjac
        for i in range(0,len(self.lis)):
            for x in range(0,len(self.lis)-1):
                if self.lis[x]>self.lis[x+1]: # ES: Si el elemento actual es mayor, intercambia /
                    aux = self.lis[x]         # ES: Guarda temporalmente el valor / EN: Temporari
                    self.lis[x] = self.lis[x+1]  # ES: Intercambia los valores / EN: Swaps the va
                    self.lis[x+1] = aux       # ES: Completa el intercambio / EN: Completes the s
        print(self.lis)                       # ES: Muestra la lista ordenada en consola / EN: Pr
        self.lista.delete(0,END)              # ES: Borra todos los elementos actuales / EN: Clear
        for i in self.lis:                    # ES: Recorre los elementos ordenados / EN: Loops th
            self.lista.insert(self.lista.size()+1,str(i))  # ES: Inserta cada elemento ordenado /

    def inicio(self):                         # ES: Configura la interfaz gráfica / EN: Sets up th
        self.dato = Entry(self.ven)           # ES: Caja de texto para ingresar datos / EN: Input
        self.dato.place(x = 50, y = 10)       # ES: Posición de la caja / EN: Entry position
        self.modo = StringVar(value="Pila")   # ES: Variable para modo (Pila o Cola) / EN: Variabl
        Radiobutton(self.ven, text="Pila",variable=self.modo,value="Pila").place(x=50,y=40)  # ES: Bot
        Radiobutton(self.ven, text="Colas",variable=self.modo,value="Colas").place(x=100,y=40) # ES: B
```

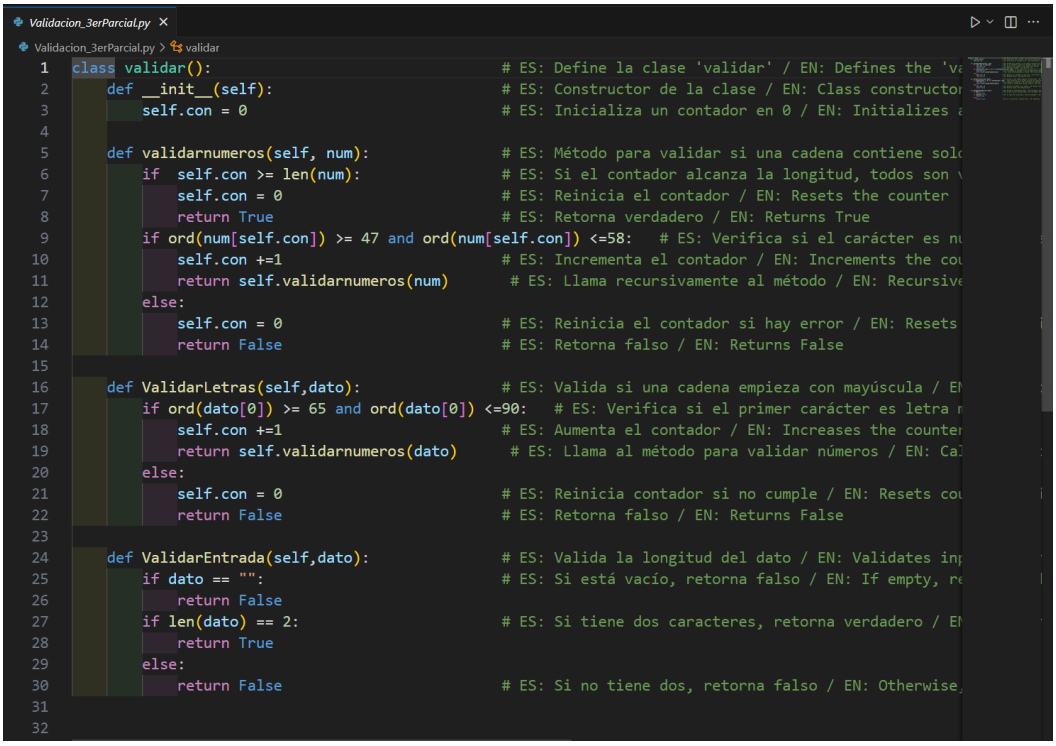
```python
class Principal():                              # ES: Define la clase Principal / EN: Defines the Pr
    def inicio(self):                         # ES: Configura la interfaz gráfica / EN: Sets up th
        Radiobutton(self.ven, text="Colas",variable=self.modo,value="Colas").place(x=100,y=40) # ES: B
        Button(self.ven, text="Validar", command=self.validarCaja).place(x = 100, y = 90)     # ES: Bo
        Button(self.ven, text="Eliminar", command=self.eliminardato).place(x = 100, y = 120)  # ES: Bo
        Button(self.ven, text="Ordendar", command=self.ordenar).place(x = 100, y = 150)       # ES: Bo
        self.label = Label(text="Numero")     # ES: Etiqueta de texto / EN: Text label
        self.label.place(x=10, y=60)          # ES: Posición de la etiqueta / EN: Label position
        self.lista = Listbox(self.ven, heig =10, width= 10, bg = "White")  # ES: Lista visual / EN: Vi
        self.lista.place (x= 210, y= 20)      # ES: Posición de la lista / EN: Listbox position
        self.ven.mainloop()                   # ES: Inicia el bucle principal de la ventana / EN:

if __name__=='__main__':                      # ES: Verifica si se ejecuta directamente / EN: Check
    app = Principal()                         # ES: Crea objeto de la clase Principal / EN: Create
    app.inicio()                              # ES: Llama al método inicio / EN: Calls the inicio
```

Ejecución

```
PS C:\Users\Carlo\OneDrive\Escritorio\Progr...on/Python313/python.exe "c:/Users/Carlo/OneDrive/Escritorio
a1_3erParcial.py"
['10', '11', '12', '22', '22']
```



Validacion programa 1



```python
class validar():                                # ES: Define la clase 'validar' / EN: Defines the 'va
    def __init__(self):                         # ES: Constructor de la clase / EN: Class constructor
        self.con = 0                            # ES: Inicializa un contador en 0 / EN: Initializes a

    def validarnumeros(self, num):              # ES: Método para validar si una cadena contiene solo
        if  self.con >= len(num):               # ES: Si el contador alcanza la longitud, todos son v
            self.con = 0                        # ES: Reinicia el contador / EN: Resets the counter
            return True                         # ES: Retorna verdadero / EN: Returns True
        if ord(num[self.con]) >= 47 and ord(num[self.con]) <=58:   # ES: Verifica si el carácter es nu
            self.con +=1                        # ES: Incrementa el contador / EN: Increments the cou
            return self.validarnumeros(num)      # ES: Llama recursivamente al método / EN: Recursive
        else:
            self.con = 0                        # ES: Reinicia el contador si hay error / EN: Resets
            return False                        # ES: Retorna falso / EN: Returns False

    def ValidarLetras(self,dato):               # ES: Valida si una cadena empieza con mayúscula / EN
        if ord(dato[0]) >= 65 and ord(dato[0]) <=90:   # ES: Verifica si el primer carácter es letra m
            self.con +=1                        # ES: Aumenta el contador / EN: Increases the counter
            return self.validarnumeros(dato)     # ES: Llama al método para validar números / EN: Cal
        else:
            self.con = 0                        # ES: Reinicia contador si no cumple / EN: Resets cou
            return False                        # ES: Retorna falso / EN: Returns False

    def ValidarEntrada(self,dato):              # ES: Valida la longitud del dato / EN: Validates inp
        if dato == "":                          # ES: Si está vacío, retorna falso / EN: If empty, re
            return False
        if len(dato) == 2:                      # ES: Si tiene dos caracteres, retorna verdadero / EN
            return True
        else:
            return False                        # ES: Si no tiene dos, retorna falso / EN: Otherwise,
```

Programa 2

```python
'''Hacer un programa que lea nombre, apellido paterno y materno en 3 cajas separadas
ademas, leer dia, mes y año en 3 cajas separadas.
Al precionar un boton se agregara a un listbox el RFC de la persona, ademas, contrendra 2
botones para eliminar elementos de listbox mediante pilas y colas'''
'''Primer letra y vocal del apeido paterno, primer letra del apeido materno, primer letra del nombre
todo mayusculas, año (2) mes (2) dia (2)'''
from tkinter import *                           # ES: Importa todas las funciones de tkinter / EN: Im
from tkinter import messagebox                  # ES: Importa los cuadros de mensajes / EN: Imports m
class Principal():                              # ES: Define la clase Principal / EN: Defines the Pri
    def __init__(self):                         # ES: Constructor de la clase / EN: Class constructor
        self.ventana = Tk()                     # ES: Crea la ventana principal / EN: Creates the ma
        ancho = 550                             # ES: Define el ancho de la ventana / EN: Sets the wi
        alto = 230                              # ES: Define la altura de la ventana / EN: Sets the v
        ventana_alto = self.ventana.winfo_screenmmwidth()   # ES: Obtiene el ancho de la pantalla en m
        ventana_ancho = self.ventana.winfo_screenmmheight() # ES: Obtiene la altura de la pantalla en
        x = (ventana_alto // 2) - (ancho // 2)   # ES: Calcula posición X centrada / EN: Calculates ce
        y = (ventana_ancho // 2) - (alto // 2)   # ES: Calcula posición Y centrada / EN: Calculates ce
        self.ventana.geometry(f"{ancho}x{alto}+{x+500}+{y+250}")  # ES: Define tamaño y posición de la
        self.lista = []                         # ES: Inicializa lista vacía para guardar RFCs / EN:
        self.con = 0                            # ES: Inicializa contador / EN: Initializes counter
    def Inicio(self):                           # ES: Crea los elementos gráficos de la interfaz / EN
        # Caja de texto / Text boxes
        self.nombre = Entry(self.ventana)       # ES: Campo de texto para nombre / EN: Text field for
        self.nombre.place(x=30,y=30)
        self.apellidoP = Entry(self.ventana)    # ES: Campo de texto para apellido paterno / EN: Text
        self.apellidoP.place(x=160,y=30)
        self.apellidoM = Entry(self.ventana)    # ES: Campo de texto para apellido materno / EN: Text
        self.apellidoM.place(x=290,y=30)
        self.dia = Entry(self.ventana)          # ES: Campo de texto para día de nacimiento / EN: Te
        self.dia.place(x=30,y=90)
        self.mes = Entry(self.ventana)          # ES: Campo de texto para mes de nacimiento / EN: Te
        self.mes.place(x=160,y=90)
```

Programa_2.py

```python
class Principal():                              # ES: Define la clase Principal / EN: Defines the Pri
    def Inicio(self):                           # ES: Crea los elementos gráficos de la interfaz / EN
        self.anio = Entry(self.ventana)         # ES: Campo de texto para año de nacimiento / EN: Tex
        self.anio.place(x=290,y=90)
        # ListBox
        self.listaBox = Listbox(self.ventana, height=13, width=20, bg='white', activestyle="dotbox", f
        # ES: Lista visual para mostrar los RFC generados / EN: Visual list to display generated RFCs
        self.listaBox.place(x=420, y=10)
        # Botones / Buttons
        Button(self.ventana,text="Agregar",command=self.agregarRFC, height=3, width=10,).place(x=50,y=
        # ES: Botón para agregar RFC / EN: Button to add RFC
        Button(self.ventana,text="Eliminar\n(Pilas)",command=self.eliminarPilas, height=3, width=12,).
        # ES: Botón para eliminar con método pila (último en entrar, primero en salir) / EN: Button to
        Button(self.ventana,text="Eliminar\n(Colas)",command=self.eliminarColas, height=3, width=12,).
        # ES: Botón para eliminar con método cola (primero en entrar, primero en salir) / EN: Button f
        # # Labels
        Label(self.ventana, text="Nombre").place(x=40, y=10)            # ES: Etiqueta para nombre /
        Label(self.ventana, text="Apellido Paterno").place(x=170, y=10)  # ES: Etiqueta para apellido
        Label(self.ventana, text="Apellido Materno").place(x=300, y=10)  # ES: Etiqueta para apellido
        Label(self.ventana, text="Dia").place(x=40, y=70)                # ES: Etiqueta para día / EN:
        Label(self.ventana, text="Mes").place(x=170, y=70)               # ES: Etiqueta para mes / EN:
        Label(self.ventana, text="Año").place(x=300, y=70)               # ES: Etiqueta para año / EN:
        # mainloop
        self.ventana.mainloop()                  # ES: Inicia el ciclo principal de la ventana / EN:
    def agregarRFC(self):                       # ES: Método para generar y agregar un RFC / EN: Meth
        # Validar nombre / Validate name
        if self.nombre.get() == "":
            messagebox.showerror("ERROR", "Escribe un Nombre")   # ES: Error si está vacío / EN: Error
            return
        else:
            if not self.validarLetras(self.nombre.get()):        # ES: Verifica que solo tenga letras
                return
```

Programa_2.py

```python
class Principal():                              # ES: Define la clase Principal / EN: Defines the Pri
    def agregarRFC(self):                       # ES: Método para generar y agregar un RFC / EN: Meth
        # Validar apellido paterno / Validate paternal surname
        if self.apellidoP.get() == "":
            messagebox.showerror("ERROR", "Escribe el apellido Paterno")
            return
        else:
            if not self.validarLetras(self.apellidoP.get()):
                return
        # Validar apellido materno / Validate maternal surname
        if self.apellidoM.get() == "":
            messagebox.showerror("ERROR", "Escribe el apellido Materno")
            return
        else:
            if not self.validarLetras(self.apellidoM.get()):
                return
        # Validar día / Validate day
        if self.dia.get() == "" or len(self.dia.get()) != 2:
            messagebox.showerror("ERROR", "Escribe un día válido de 2 dígitos")
            return
        else:
            if not self.validarNumeros(self.dia.get()):
                return
        # Validar mes / Validate month
        if self.mes.get() == "" or len(self.mes.get()) != 2:
            messagebox.showerror("ERROR", "Escribe un mes válido de 2 dígitos")
            return
        else:
            if not self.validarNumeros(self.mes.get()):
                return
        # Validar año / Validate year
        if self.anio.get() == "" or len(self.anio.get()) != 4:
```

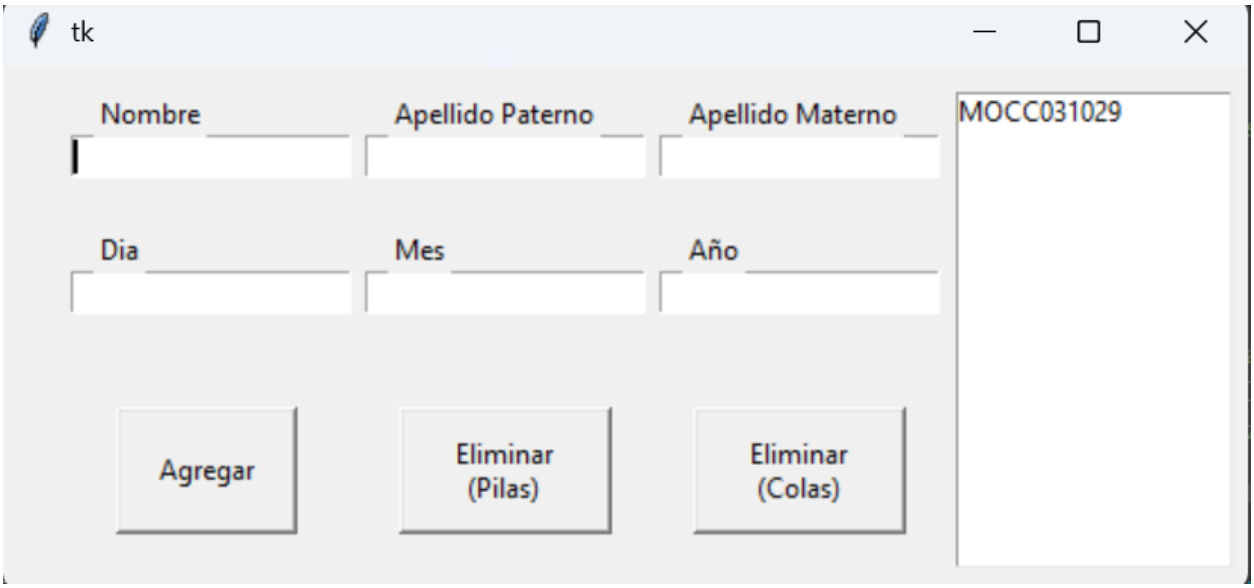
Programa_2.py

```
 9  class Principal():                              # ES: Define la clase Principal / EN: Defines the Pr
55      def agregarRFC(self):                       # ES: Método para generar y agregar un RFC / EN: Met
93              messagebox.showerror("ERROR", "Escribe un año válido de 4 dígitos")
94              return
95          else:
96              if not self.validarNumeros(self.anio.get()):
97                  return
98          # Construir RFC / Build RFC
99          self.rfc = ""
100         self.rfc = (
101             f"{self.apellidoP.get().upper()[:2]}"  # ES: Primeras dos letras del apellido paterno / EN
102             f"{self.apellidoM.get().upper()[0]}"   # ES: Primera letra del apellido materno / EN: Firs
103             f"{self.nombre.get().upper()[0]}"      # ES: Primera letra del nombre / EN: First letter o
104             f"{self.anio.get()[2:4]}"              # ES: Últimos dos dígitos del año / EN: Last two di
105             f"{self.mes.get()}"                    # ES: Mes completo / EN: Full month
106             f"{self.dia.get()}"                    # ES: Día completo / EN: Full day
107         )
108         self.lista.append(self.rfc)                # ES: Agrega RFC a la lista / EN: Adds RFC to inter
109         self.listaBox.insert(self.listaBox.size()+1,self.rfc)  # ES: Muestra RFC en la lista visual /
110         # Limpia los campos / Clears all fields
111         self.nombre.delete(0,END)
112         self.apellidoM.delete(0,END)
113         self.apellidoP.delete(0,END)
114         self.dia.delete(0,END)
115         self.mes.delete(0,END)
116         self.anio.delete(0,END)
117     def eliminarPilas(self):                       # ES: Elimina último RFC (modo pila) / EN: Deletes l
118         if self.listaBox.size() <= 0:              # ES: Verifica si la lista está vacía / EN: Checks i
119             messagebox.showerror("ERROR", "La lista esta vacia")
120             return
121         else:
122             self.listaBox.delete(self.listaBox.size()-1)  # ES: Elimina último elemento / EN: Deletes
```

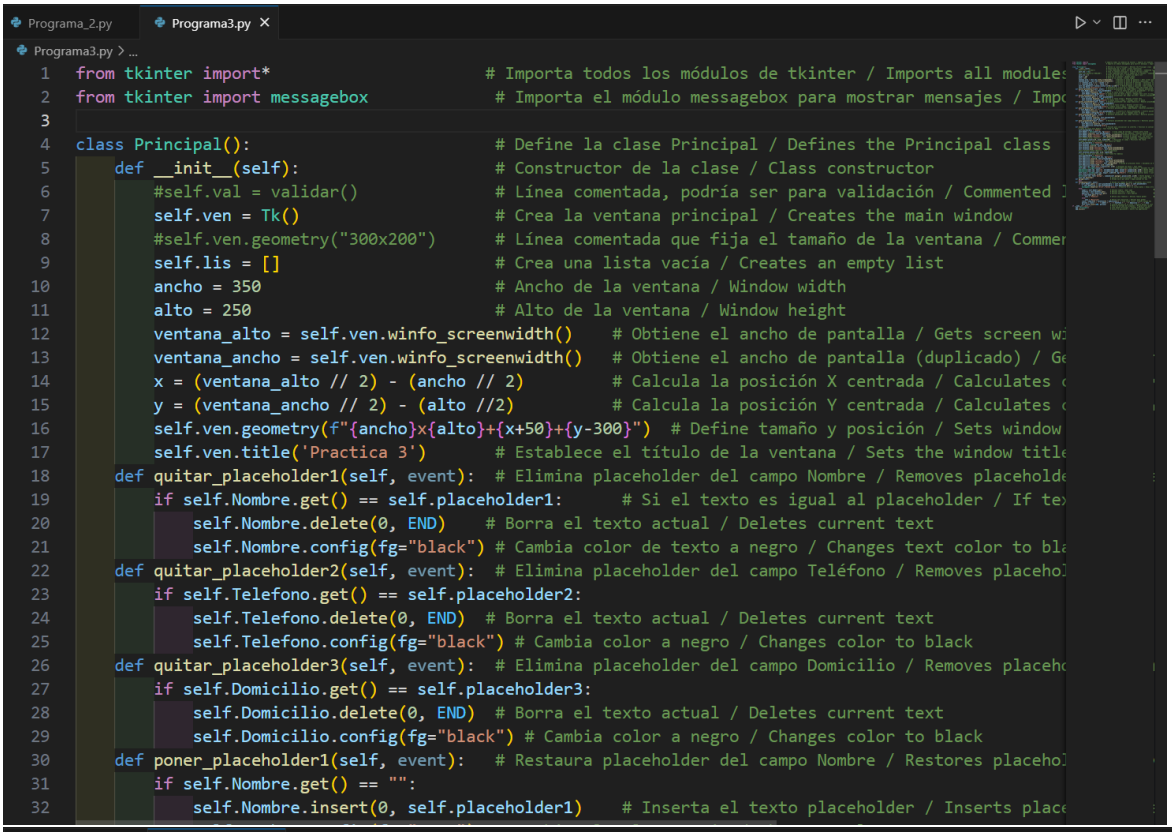
Programa_2.py

```
 9  class Principal():                              # ES: Define la clase Principal / EN: Defines the Pr
123     def eliminarColas(self):                    # ES: Elimina primer RFC (modo cola) / EN: Deletes f
124         if self.listaBox.size() <= 0:
125             messagebox.showerror("ERROR", "La lista esta vacia")
126             return
127         else:
128             self.listaBox.delete(0)
129     def validarLetras(self, dato):              # ES: Valida que el dato contenga solo letras / EN:
130         if dato.isalpha():
131             return True
132         else:
133             messagebox.showerror("ERROR", f"( {dato} ) contiene numeros.")  # ES: Muestra error si cor
134             return False
135     def validarNumeros(self, numero):           # ES: Valida que el dato contenga solo números / EN:
136         if numero.isdigit():
137             return True
138         else:
139             messagebox.showerror("ERROR", f"( {numero} ) contiene letras.") # ES: Muestra error si cor
140             return False
141 if __name__=='__main__':                        # ES: Verifica si el archivo se ejecuta directamente
142     app = Principal()                           # ES: Crea una instancia de la clase Principal / EN:
143     app.Inicio()                                # ES: Llama al método para iniciar la interfaz / EN:
```

Ejecución


tk
Nombre
Apellido Paterno
Apellido Materno
MOCC031029
Dia
Mes
Año
Agregar
Eliminar (Pilas)
Eliminar (Colas)

Programa 3

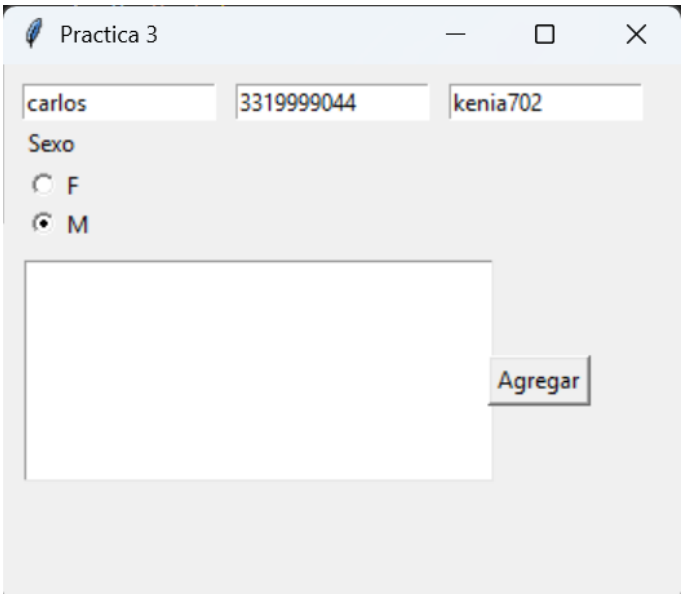


```python
from tkinter import*                    # Importa todos los módulos de tkinter / Imports all modules
from tkinter import messagebox          # Importa el módulo messagebox para mostrar mensajes / Impo

class Principal():                      # Define la clase Principal / Defines the Principal class
    def __init__(self):                 # Constructor de la clase / Class constructor
        #self.val = validar()            # Línea comentada, podría ser para validación / Commented
        self.ven = Tk()                 # Crea la ventana principal / Creates the main window
        #self.ven.geometry("300x200")    # Línea comentada que fija el tamaño de la ventana / Commer
        self.lis = []                   # Crea una lista vacía / Creates an empty list
        ancho = 350                     # Ancho de la ventana / Window width
        alto = 250                      # Alto de la ventana / Window height
        ventana_alto = self.ven.winfo_screenwidth()    # Obtiene el ancho de pantalla / Gets screen wi
        ventana_ancho = self.ven.winfo_screenwidth()   # Obtiene el ancho de pantalla (duplicado) / Ge
        x = (ventana_alto // 2) - (ancho // 2)         # Calcula la posición X centrada / Calculates c
        y = (ventana_ancho // 2) - (alto //2)          # Calcula la posición Y centrada / Calculates c
        self.ven.geometry(f"{ancho}x{alto}+{x+50}+{y-300}")  # Define tamaño y posición / Sets window
        self.ven.title('Practica 3')    # Establece el título de la ventana / Sets the window title
    def quitar_placeholder1(self, event):  # Elimina placeholder del campo Nombre / Removes placeholde
        if self.Nombre.get() == self.placeholder1:      # Si el texto es igual al placeholder / If te
            self.Nombre.delete(0, END)    # Borra el texto actual / Deletes current text
            self.Nombre.config(fg="black") # Cambia color de texto a negro / Changes text color to bla
    def quitar_placeholder2(self, event):  # Elimina placeholder del campo Teléfono / Removes placeho
        if self.Telefono.get() == self.placeholder2:
            self.Telefono.delete(0, END)  # Borra el texto actual / Deletes current text
            self.Telefono.config(fg="black") # Cambia color a negro / Changes color to black
    def quitar_placeholder3(self, event):  # Elimina placeholder del campo Domicilio / Removes placeho
        if self.Domicilio.get() == self.placeholder3:
            self.Domicilio.delete(0, END)  # Borra el texto actual / Deletes current text
            self.Domicilio.config(fg="black") # Cambia color a negro / Changes color to black
    def poner_placeholder1(self, event):   # Restaura placeholder del campo Nombre / Restores placeho
        if self.Nombre.get() == "":
            self.Nombre.insert(0, self.placeholder1)    # Inserta el texto placeholder / Inserts place
```

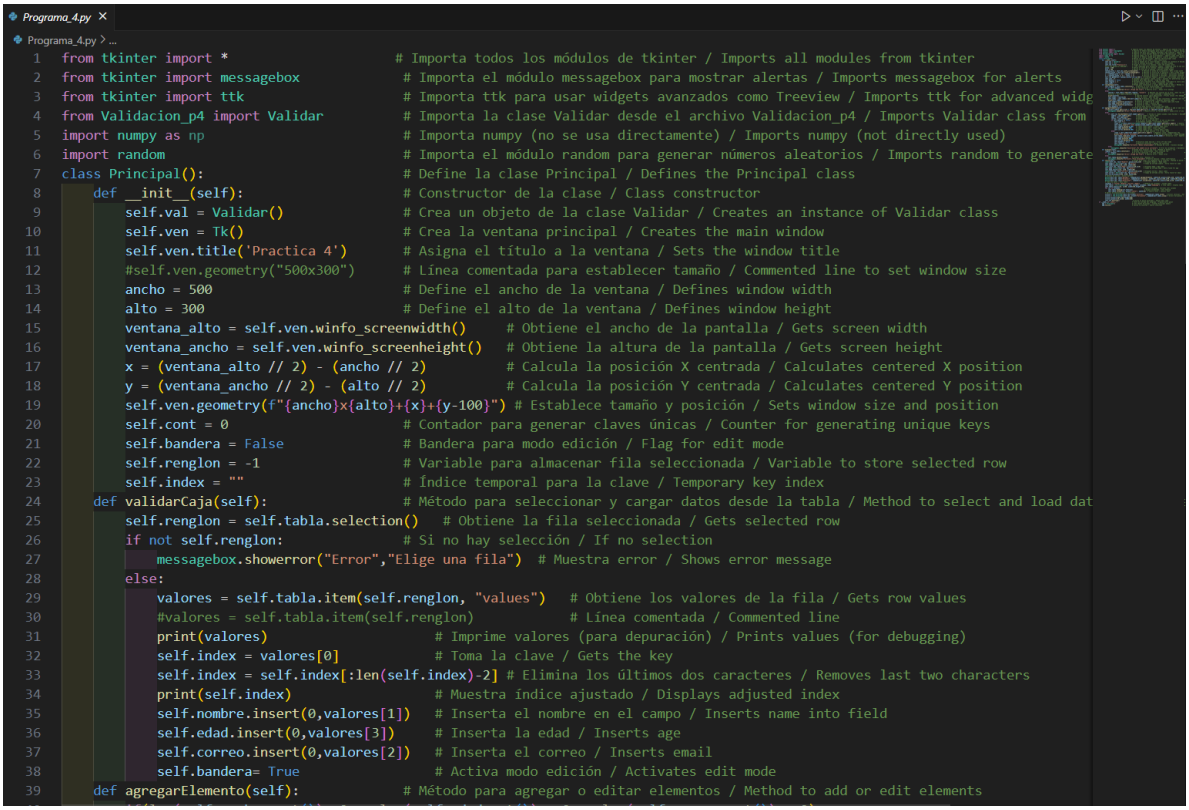
```python
class Principal():                      # Define la clase Principal / Defines the Principal class
    def poner_placeholder1(self, event):   # Restaura placeholder del campo Nombre / Restores placeho
            self.Nombre.config(fg="gray")  # Cambia el color a gris / Changes color to gray
    def poner_placeholder2(self, event):   # Restaura placeholder del campo Teléfono / Restores placeh
        if self.Telefono.get() == "":
            self.Telefono.insert(0, self.placeholder2)
            self.Telefono.config(fg="gray")
    def poner_placeholder3(self, event):   # Restaura placeholder del campo Domicilio / Restores place
        if self.Domicilio.get() == "":
            self.Domicilio.insert(0, self.placeholder3)
            self.Domicilio.config(fg="gray")
    def inicio(self):                      # Función para inicializar la interfaz / Function to initia
        # Campo de texto para Nombre / Text field for Name
        self.placeholder1 = "Nombre"
        self.Nombre = Entry(self.ven, fg="gray")   # Crea campo de entrada / Creates entry field
        self.Nombre.insert(0, self.placeholder1)   # Inserta texto por defecto / Inserts default text
        self.Nombre.bind("<FocusIn>", self.quitar_placeholder1) # Evento al enfocar / On focus event
        self.Nombre.bind("<FocusOut>", self.poner_placeholder1) # Evento al perder foco / On focus out
        #self.Nombre.bind("<Return>", self.validarCaja)          # Línea comentada / Commented line
        self.Nombre.place(x=10, y=10, width=100)   # Posiciona el campo / Positions the field
        # Campo de texto para Teléfono / Text field for Phone
        self.placeholder2 = "Telefono"
        self.Telefono = Entry(self.ven, fg="gray")
        self.Telefono.insert(0, self.placeholder2)
        self.Telefono.bind("<FocusIn>", self.quitar_placeholder2)
        self.Telefono.bind("<FocusOut>", self.poner_placeholder2)
        #self.Telefono.bind("<Return>", self.validarCaja)
        self.Telefono.place(x=120, y=10, width=100)
        # Campo de texto para Domicilio / Text field for Address
        self.placeholder3 = "Domicilio"
        self.Domicilio = Entry(self.ven, fg="gray")
        self.Domicilio.insert(0, self.placeholder3)
```

```
Programa_2.py   Programa3.py
Programa3.py > ...
  5  class Principal():                          # Define la clase Principal / Defines the Principal class
 43      def inicio(self):                       # Función para inicializar la interfaz / Function to initialize the interface
 63          self.Domicilio.insert(0, self.placeholder3)
 64          self.Domicilio.bind("<FocusIn>", self.quitar_placeholder3)
 65          self.Domicilio.bind("<FocusOut>", self.poner_placeholder3)
 66          self.Domicilio.bind("<Return>", self.validarCaja)  # Valida al presionar Enter / Validates on Enter
 67          self.Domicilio.place(x=230, y=10, width=100)
 68          Label(self.ven, text='Sexo').place(x=10, y=30)  # Etiqueta de texto / Text label
 69          self.modo = StringVar(value='F')            # Variable para radio buttons / Variable for radio buttons
 70          Radiobutton(self.ven, text="F", variable=self.modo, value="F").place(x=10, y=50) # Botón femenino / Female button
 71          Radiobutton(self.ven, text="M", variable=self.modo, value="M").place(x=10, y=70) # Botón masculino / Male button
 72          self.lista = Listbox(self.ven, heig=7, width=40, bg="White")  # Lista para mostrar datos / Listbox to show data
 73          self.lista.place(x=10, y=100)
 74          Button(self.ven, text='Agregar', command=self.agregar).place(x=250, y=150)  # Botón Agregar / Add button
 75          self.ven.mainloop()                     # Inicia el bucle principal / Starts main event loop
 76      def agregar(self):                          # Método vacío por ahora / Empty method for now
 77          pass
 78      def validarCaja(self, event):               # Valida los campos de texto / Validates text fields
 79          if (self.Nombre.get() == self.placeholder1 or self.Telefono.get() == self.placeholder2
 80              or self.Domicilio == self.placeholder3 or self.Domicilio.get() == ""):
 81              messagebox.showerror('Error','Faltan datos')   # Muestra error si faltan datos / Shows error if data missing
 82          else:
 83              nombre = self.Nombre.get()          # Obtiene nombre / Gets name
 84              telefono = self.Telefono.get()      # Obtiene teléfono / Gets phone
 85              domicilio = self.Domicilio.get()    # Obtiene domicilio / Gets address
 86              if self.modo.get() == "F":
 87                  sexo = "Femenino"               # Define sexo femenino / Defines female gender
 88              else:
 89                  sexo = "Masculino"              # Define sexo masculino / Defines male gender
 90              clave = nombre[0] + telefono[0] + domicilio[2:]  # Crea clave única / Creates unique key
 91              persona = clave + " _" + nombre + " _ " + telefono + " _ " + domicilio + " _ " + sexo
 92              self.lista.insert(END, persona)     # Inserta persona en la lista / Inserts person in list
 93  if __name__=='__main__':                        # Punto de entrada del programa / Program entry point
 94      app = Principal()                           # Crea objeto Principal / Creates Principal object
 95      app.inicio()                                # Inicia la aplicación / Starts the application
```

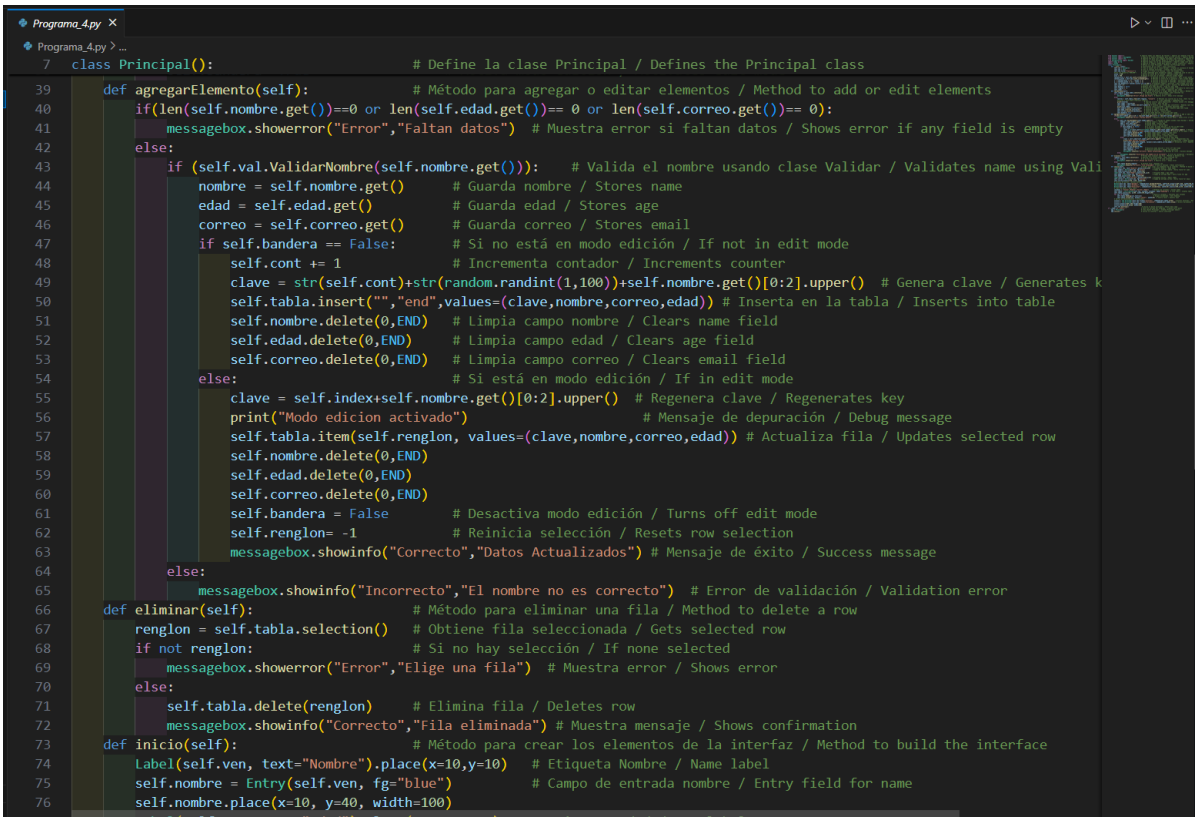
Ejecución



Programa 4



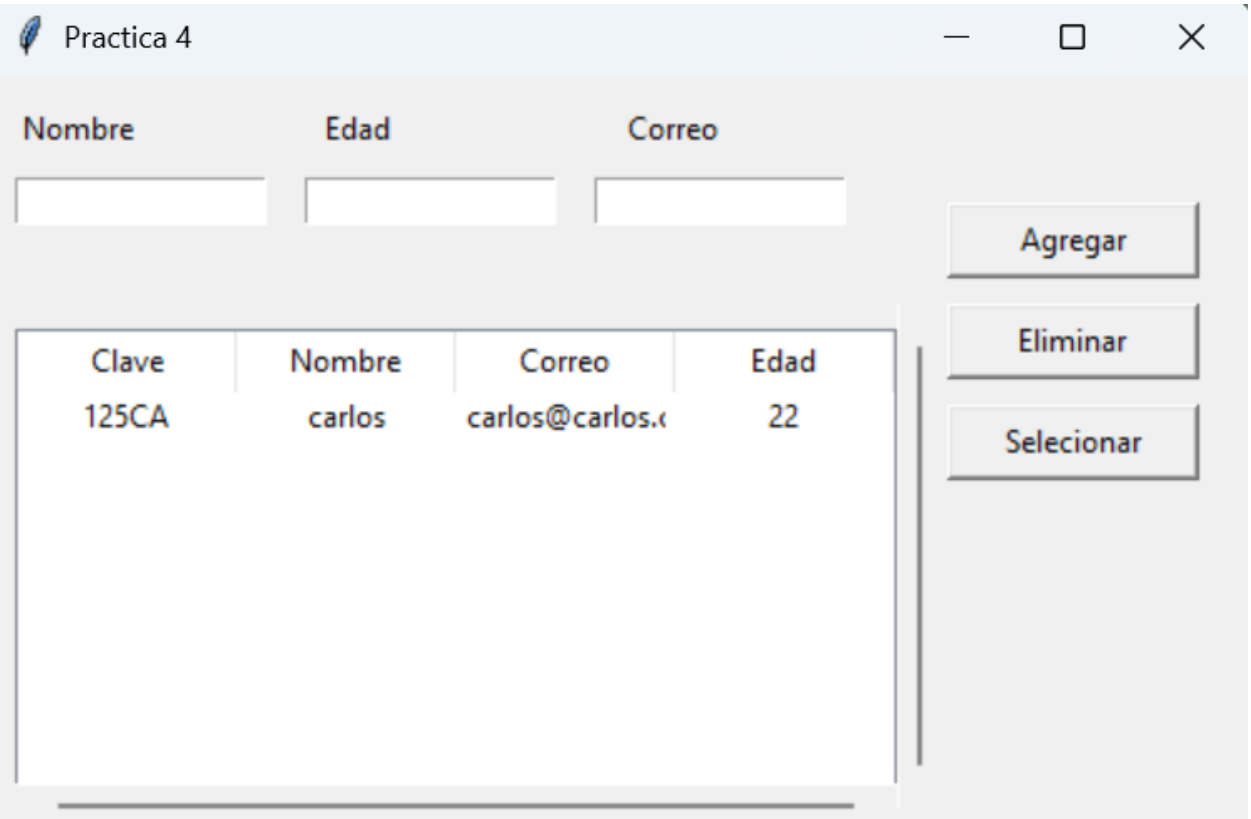
```python
from tkinter import *                    # Importa todos los módulos de tkinter / Imports all modules from tkinter
from tkinter import messagebox           # Importa el módulo messagebox para mostrar alertas / Imports messagebox for alerts
from tkinter import ttk                  # Importa ttk para usar widgets avanzados como Treeview / Imports ttk for advanced widg
from Validacion_p4 import Validar        # Importa la clase Validar desde el archivo Validacion_p4 / Imports Validar class from
import numpy as np                       # Importa numpy (no se usa directamente) / Imports numpy (not directly used)
import random                            # Importa el módulo random para generar números aleatorios / Imports random to generate
class Principal():                       # Define la clase Principal / Defines the Principal class
    def __init__(self):                  # Constructor de la clase / Class constructor
        self.val = Validar()             # Crea un objeto de la clase Validar / Creates an instance of Validar class
        self.ven = Tk()                  # Crea la ventana principal / Creates the main window
        self.ven.title('Practica 4')     # Asigna el título a la ventana / Sets the window title
        #self.ven.geometry("500x300")     # Línea comentada para establecer tamaño / Commented line to set window size
        ancho = 500                      # Define el ancho de la ventana / Defines window width
        alto = 300                       # Define el alto de la ventana / Defines window height
        ventana_alto = self.ven.winfo_screenwidth()     # Obtiene el ancho de la pantalla / Gets screen width
        ventana_ancho = self.ven.winfo_screenheight()   # Obtiene la altura de la pantalla / Gets screen height
        x = (ventana_alto // 2) - (ancho // 2)          # Calcula la posición X centrada / Calculates centered X position
        y = (ventana_ancho // 2) - (alto // 2)          # Calcula la posición Y centrada / Calculates centered Y position
        self.ven.geometry(f"{ancho}x{alto}+{x}+{y-100}") # Establece tamaño y posición / Sets window size and position
        self.cont = 0                    # Contador para generar claves únicas / Counter for generating unique keys
        self.bandera = False             # Bandera para modo edición / Flag for edit mode
        self.renglon = -1                # Variable para almacenar fila seleccionada / Variable to store selected row
        self.index = ""                  # Índice temporal para la clave / Temporary key index
    def validarCaja(self):               # Método para seleccionar y cargar datos desde la tabla / Method to select and load dat
        self.renglon = self.tabla.selection()   # Obtiene la fila seleccionada / Gets selected row
        if not self.renglon:             # Si no hay selección / If no selection
            messagebox.showerror("Error","Elige una fila")  # Muestra error / Shows error message
        else:
            valores = self.tabla.item(self.renglon, "values")   # Obtiene los valores de la fila / Gets row values
            #valores = self.tabla.item(self.renglon)            # Línea comentada / Commented line
            print(valores)                   # Imprime valores (para depuración) / Prints values (for debugging)
            self.index = valores[0]          # Toma la clave / Gets the key
            self.index = self.index[:len(self.index)-2] # Elimina los últimos dos caracteres / Removes last two characters
            print(self.index)                # Muestra índice ajustado / Displays adjusted index
            self.nombre.insert(0,valores[1])   # Inserta el nombre en el campo / Inserts name into field
            self.edad.insert(0,valores[3])     # Inserta la edad / Inserts age
            self.correo.insert(0,valores[2])   # Inserta el correo / Inserts email
            self.bandera= True               # Activa modo edición / Activates edit mode
    def agregarElemento(self):           # Método para agregar o editar elementos / Method to add or edit elements
```



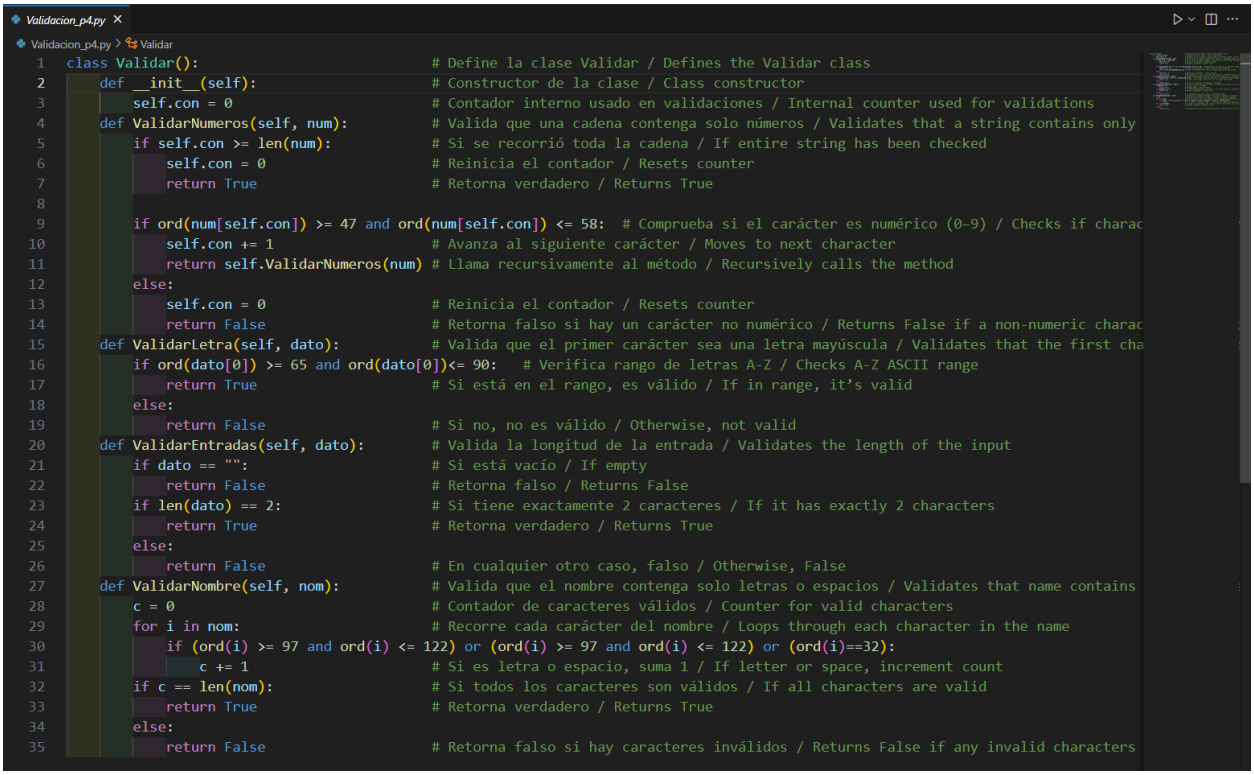
```python
class Principal():                       # Define la clase Principal / Defines the Principal class
    def agregarElemento(self):           # Método para agregar o editar elementos / Method to add or edit elements
        if(len(self.nombre.get())==0 or len(self.edad.get())== 0 or len(self.correo.get())== 0):
            messagebox.showerror("Error","Faltan datos")  # Muestra error si faltan datos / Shows error if any field is empty
        else:
            if (self.val.ValidarNombre(self.nombre.get())):    # Valida el nombre usando clase Validar / Validates name using Vali
                nombre = self.nombre.get()      # Guarda nombre / Stores name
                edad = self.edad.get()          # Guarda edad / Stores age
                correo = self.correo.get()      # Guarda correo / Stores email
                if self.bandera == False:       # Si no está en modo edición / If not in edit mode
                    self.cont += 1              # Incrementa contador / Increments counter
                    clave = str(self.cont)+str(random.randint(1,100))+self.nombre.get()[0:2].upper()  # Genera clave / Generates k
                    self.tabla.insert("","end",values=(clave,nombre,correo,edad)) # Inserta en la tabla / Inserts into table
                    self.nombre.delete(0,END)   # Limpia campo nombre / Clears name field
                    self.edad.delete(0,END)     # Limpia campo edad / Clears age field
                    self.correo.delete(0,END)   # Limpia campo correo / Clears email field
                else:                           # Si está en modo edición / If in edit mode
                    clave = self.index+self.nombre.get()[0:2].upper()  # Regenera clave / Regenerates key
                    print("Modo edicion activado")                     # Mensaje de depuración / Debug message
                    self.tabla.item(self.renglon, values=(clave,nombre,correo,edad)) # Actualiza fila / Updates selected row
                    self.nombre.delete(0,END)
                    self.edad.delete(0,END)
                    self.correo.delete(0,END)
                    self.bandera = False        # Desactiva modo edición / Turns off edit mode
                    self.renglon= -1            # Reinicia selección / Resets row selection
                    messagebox.showinfo("Correcto","Datos Actualizados") # Mensaje de éxito / Success message
            else:
                messagebox.showinfo("Incorrecto","El nombre no es correcto")  # Error de validación / Validation error
    def eliminar(self):                  # Método para eliminar una fila / Method to delete a row
        renglon = self.tabla.selection()   # Obtiene fila seleccionada / Gets selected row
        if not renglon:                  # Si no hay selección / If none selected
            messagebox.showerror("Error","Elige una fila")  # Muestra error / Shows error
        else:
            self.tabla.delete(renglon)     # Elimina fila / Deletes row
            messagebox.showinfo("Correcto","Fila eliminada") # Muestra mensaje / Shows confirmation
    def inicio(self):                    # Método para crear los elementos de la interfaz / Method to build the interface
        Label(self.ven, text="Nombre").place(x=10,y=10)   # Etiqueta Nombre / Name label
        self.nombre = Entry(self.ven, fg="blue")          # Campo de entrada nombre / Entry field for name
        self.nombre.place(x=10, y=40, width=100)
```

```python
class Principal():                      # Define la clase Principal / Defines the Principal class
    def inicio(self):                   # Método para crear los elementos de la interfaz / Method to build the interface
        Label(self.ven, text="Edad").place(x=130,y=10)    # Etiqueta Edad / Age label
        self.edad = Entry(self.ven, fg="green")           # Campo de entrada edad / Entry field for age
        self.edad.place(x=125, y=40, width=100)
        Label(self.ven, text="Correo").place(x=250,y=10)  # Etiqueta Correo / Email label
        self.correo = Entry(self.ven, fg="purple")        # Campo de entrada correo / Entry field for email
        self.correo.place(x=240, y=40, width=100)
        # Botones de acción / Action buttons
        Button(self.ven, text="Agregar", command=self.agregarElemento, width=10).place(x=380,y=50, width=100,height=30)
        Button(self.ven, text="Eliminar", command=self.eliminar, width=10).place(x=380,y=90, width=100,height=30)
        Button(self.ven, text="Selecionar", command=self.validarCaja, width=10).place(x=380,y=130, width=100,height=30)
        # Tabla de datos / Data table
        columnas = ("Clave","Nombre","Correo","Edad")     # Nombres de columnas / Column names
        self.tabla = ttk.Treeview(self.ven, columns= columnas, show="headings")  # Crea tabla / Creates table
        self.tabla.place(x=10, y=100, width=350,height=190)
        for col in columnas:                              # Recorre columnas / Iterates over columns
            self.tabla.heading(col,text=col)              # Asigna encabezado / Sets heading
            self.tabla.column(col, anchor="center", width=30)  # Centra texto / Centers text
        # Barras de desplazamiento / Scrollbars
        scrolly = ttk.Scrollbar(self.ven,orient="vertical", command=self.tabla.yview)   # Scroll vertical / Vertical scroll
        scrollx = ttk.Scrollbar(self.ven, orient="horizontal", command=self.tabla.xview) # Scroll horizontal / Horizontal scroll
        scrolly.place(x=360,y=90,height=200)
        scrollx.place(x=10,y=280, width=350)
        self.ven.mainloop()                # Inicia el bucle principal / Starts main loop
if __name__=='__main__':                   # Punto de entrada del programa / Program entry point
    app = Principal()                      # Crea objeto Principal / Creates Principal object
    app.inicio()                           # Inicia la interfaz / Starts interface
```

Ejecución



Validacion progrma 4



```python
class Validar():                        # Define la clase Validar / Defines the Validar class
    def __init__(self):                 # Constructor de la clase / Class constructor
        self.con = 0                    # Contador interno usado en validaciones / Internal counter used for validations
    def ValidarNumeros(self, num):      # Valida que una cadena contenga solo números / Validates that a string contains only
        if self.con >= len(num):        # Si se recorrió toda la cadena / If entire string has been checked
            self.con = 0                # Reinicia el contador / Resets counter
            return True                 # Retorna verdadero / Returns True

        if ord(num[self.con]) >= 47 and ord(num[self.con]) <= 58:  # Comprueba si el carácter es numérico (0-9) / Checks if charac
            self.con += 1               # Avanza al siguiente carácter / Moves to next character
            return self.ValidarNumeros(num) # Llama recursivamente al método / Recursively calls the method
        else:
            self.con = 0                # Reinicia el contador / Resets counter
            return False                # Retorna falso si hay un carácter no numérico / Returns False if a non-numeric charac
    def ValidarLetra(self, dato):       # Valida que el primer carácter sea una letra mayúscula / Validates that the first cha
        if ord(dato[0]) >= 65 and ord(dato[0])<= 90:   # Verifica rango de letras A-Z / Checks A-Z ASCII range
            return True                 # Si está en el rango, es válido / If in range, it's valid
        else:
            return False                # Si no, no es válido / Otherwise, not valid
    def ValidarEntradas(self, dato):    # Valida la longitud de la entrada / Validates the length of the input
        if dato == "":                  # Si está vacío / If empty
            return False                # Retorna falso / Returns False
        if len(dato) == 2:              # Si tiene exactamente 2 caracteres / If it has exactly 2 characters
            return True                 # Retorna verdadero / Returns True
        else:
            return False                # En cualquier otro caso, falso / Otherwise, False
    def ValidarNombre(self, nom):       # Valida que el nombre contenga solo letras o espacios / Validates that name contains
        c = 0                           # Contador de caracteres válidos / Counter for valid characters
        for i in nom:                   # Recorre cada carácter del nombre / Loops through each character in the name
            if (ord(i) >= 97 and ord(i) <= 122) or (ord(i) >= 97 and ord(i) <= 122) or (ord(i)==32):
                c += 1                  # Si es letra o espacio, suma 1 / If letter or space, increment count
        if c == len(nom):               # Si todos los caracteres son válidos / If all characters are valid
            return True                 # Retorna verdadero / Returns True
        else:
            return False                # Retorna falso si hay caracteres inválidos / Returns False if any invalid characters
```