2022

Manual Técnico

MARIO AARON LOPEZ JOAQUIN 202109554 LFP A+

Paradigma Aplicado de Programación

Para esta práctica se implementó Programación Orientada a Objetos debido a que se instancia una clase llamada Cursos con sus atributos: código, nombre, pre-requisito, semestre, opcionalidad, créditos y estado. Este objeto servirá para poder almacenar todos los registros de cursos para la carrera de Ingeniería en Ciencias y Sistemas.

Convenciones de Nomenclatura

Class Cursos(): Esta clase es el objeto para los cursos de la carrera. Dentro de ella se definen todos los parámetros de cada curso, es decir sus atributos. Esta clase es esencial para poder ingresar los cursos ya que dentro de ella se define una lista que inicialmente estará vacía y se irán agregando los cursos con todos sus atributos.

Ingreso_registro(): Esta función añade a una tabla (Treeview en Tkinter) todos los cursos que se han añadido a la lista definida en la clase cursos.

Agregar_curso(): Esta función se encarga de recibir todos los datos que ingresó el usuario para crear un nuevo curso que se añadirá a la lista de cursos existentes.

Editar_curso(): Esta función se encarga de editar un curso especifico que será definido por el usuario que ya existe en el listado de cursos.

Eliminar_curso(): Esta función eliminará el curso que el usuario especifique por completo sin dejar ninguno de sus atributos dentro de la lista.

Buscar_curso(): Esta función buscará dentro del listado de cursos un curso específico en base a su código y lo regresará para poder realizar la actualización de cursos.

Conteo_creditos_aprobados(): Esta función buscará dentro del listado de cursos todos los que han sido aprobados y sumará los créditos de cada curso.

Conteo_creditos_cursando(): Esta función buscará dentro del listado de cursos todos los que tienen como estado “cursando” y sumará los créditos de cada curso.

Conteo_creditos_pendientes(): Esta función buscará dentro del listado de cursos todos los que están pendiente y tienen como opcionalidad “obligatorio” y sumará los créditos de cada curso.

Creditos_obligatorios(): Esta función buscará dentro del listado de cursos todos los que son del semestre que el usuario especifico y de semestres anteriores para sumar los créditos de todos los cursos obligatorios.

Creditos_semestre(): Esta función buscará dentro del listado de cursos todos los que son de un semestre especificado por el usuario para devolver la suma de los créditos de cada curso de dicho semestre.

Métodos Principales

Clase Menu Principal

```
class menu(Tk):
    def __init__(self):
        super().__init__()
        self.title("Practica 1")
        self.geometry("500x250")
        self.resizable(False, False)
        self.configure(bg = "skyblue")
```

Para crear el menú principal se usó una clase que recibe como parámetro Tk que hace referencia a que se estará trabajando con la librería tkinter dentro de la clase.

La función __init__ es el constructor de la clase y solamente recibirá como parámetro el self que hace referencia a todas las variables que existirán únicamente dentro de la clase. Se definen también las propiedades de la ventana como sus dimensiones de espacio y el color de fondo que serán visibles al momento de ejecutar el programa.

```
curso = Label(text = "Nombre del curso: Lab. Lenguajes Formales y de Programacion")
curso.config(font = ("Arial", 10 , "bold"))
curso.place(x = 50, y = 5)
nombre = Label(text = 'Nombre: Mario Aaron Lopez Joaquin')
nombre.config(font = ("Arial", 10 , "bold"))
nombre.place( x = 50, y = 30)
carnet = Label(text = "Carnet: 202109554")
carnet.config(font = ("Arial", 10 , "bold"))
carnet.place(x = 50, y = 55)
```

Dentro de la clase se crearán todas las herramientas necesarias para que el usuario realice el manejo de datos. Dentro de esta ventana solamente estarán los botones para realizar el cambio de ventana.

```
    def cargar_archivo():
        ventana2 = seleccion()
        ventana2.mainloop()
    btnCarga = Button(self, text = "Cargar Archivo", command = cargar_archivo)
    btnCarga.place(x = 205, y = 80)
```

Para poder realizar el cambio de ventana se creará la función que en este caso se llama cargar_archivo en la cual se abrirá una nueva clase llamada selección la cual contiene

diferentes elementos. Luego esta función se agrega a un botón para que se ejecute al momento que el usuario presione dicho botón.

```
    def salir():
        self.destroy()

    btnSalir = Button(text = "Salir", command = salir)
    btnSalir.place(x = 220, y = 200)
```

Para salir de la ventana se llama al método destroy que en este caso está contenido en una función llamada salir que a su vez está instanciada en un botón. Al momento que el usuario presione este botón se cerrara la ventana por lo que el sistema dejaría de ejecutarse.

```
if __name__ == "__main__":
    app = menu()
    app.mainloop()
```

Este if servirá para definir la clase menú como la clase principal y está será la que se cargará por defecto al ejecutar el programa.

Clase Selección

```
from tkinter import*
from tkinter import filedialog
from tkinter import messagebox
from listacursos import listacursos
from cursos import Cursos
class seleccion(Tk):
    def __init__(self):
        super().__init__()
        self.title("Seleccionar Archivo")
        self.geometry("400x150")
        self.config(bg = "skyblue")
        self.resizable(False, False)

        LRuta = Label(self, text = "Ruta")
        LRuta.place(x = 50, y = 50)
        txtruta = StringVar()
        Ruta = Entry(self, textvariable = txtruta, width = "30")
        Ruta.place ( x = 150, y = 50)
```

Se define la clase selección con su método constructor y se definen también las herramientas que poseerá la nueva ventana. Asimismo, se toma como parámetro Tk para indicar que dentro de esta clase también se trabajará con tkinter.

```
        def seleccion_archivo():
            archivo = filedialog.askopenfilename(title = "abrir", initialdir = "C:/", filetypes=(("Archivos
csv", "*.csv"), ("Archivos LFP", "*.lfp")))
            f = open(archivo, 'r', encoding="utf-8", errors = "ignore")
            try:
                print ("Archivo Seleccionado Correctamente")
                for linea in f:
                    separacion = linea.split(",")
                    for curso in Cursos.listadoCursos:
                        if separacion[0].strip() == curso[0].strip():
                            Cursos.listadoCursos.remove(curso)
                    print(separacion)
                    Cursos.listadoCursos.append(separacion)
                messagebox.showinfo("Carga","Cursos Cargados Correctamente")
            except:
                print("Archivo no Encontrado")
            finally:
                f.close()
                regresar()
        btnSeleccionar = Button(self, text = "Seleccionar", command = seleccion_archivo)
```

La función selección_archivo creara un filedialog para que el usuario selección un archivo de su explorador de archivos el cual según lo definido en el filetypes solamente será de tipo csv o lfp. Una vez se escoja el archivo este se abrirá y se utilizará el encoding utf-8 para evitar que de problemas con las tildes al momento de que se lea el archivo línea por línea. Para poder guardar cada línea como una lista primero se debe de iterar por medio de un for línea in f que ira de línea en línea. Luego se usa el método **Split** para que se separen los elementos que están entre comas en una lista. Finalmente se realiza una validación para verificar que no hayan cursos repetidos en el archivo por medio del método **strip** que devuelve un valor acorde a lo que se quiere analizar (en este caso al código de curso que haya en cada fila del archivo).

Clase Lista Cursos

```
from tkinter import *
from tkinter.ttk import Treeview
from cursos import Cursos
from registrocursos import registrocursos
class listacursos(Tk):
    def __init__(self):
        super().__init__()
        self.title("Listar Cursos")
        self.geometry("900x500")
        self.config(bg = "skyblue")
        self.resizable(False, False)
```

Se define la clase llamada listacursos con un parámetro Tk que permitirá el uso de tkinter dentro de ella. Asimismo, se definen las cualidades de la ventana que se observarán al momento de ejecutar el programa.

```
        tabla = Treeview(self)
        tabla['columns'] = ('Codigo', 'Nombre', 'Pre_requisitos', 'Opcionalidad', 'Semestre', 'Creditos',
'Estado')
        tabla.column("#0", width = 0, stretch= NO)
        tabla.column("Codigo", anchor = CENTER, width = 80)
        tabla.column("Nombre", anchor = CENTER, width = 260)
        tabla.column("Pre_requisitos", anchor = CENTER, width = 120)
        tabla.column("Opcionalidad", anchor = CENTER, width = 80)
        tabla.column("Semestre", anchor = CENTER, width = 80)
        tabla.column("Creditos", anchor = CENTER, width = 80)
        tabla.column("Estado", anchor = CENTER, width = 160)
        tabla.heading("#0", text = "", anchor = CENTER)
        tabla.heading("Codigo", text = "Codigo", anchor = CENTER)
        tabla.heading("Nombre", text = "Nombre", anchor = CENTER)
        tabla.heading("Pre_requisitos", text = "Pre requisitos", anchor = CENTER)
        tabla.heading("Opcionalidad", text = "Opcionalidad", anchor = CENTER)
        tabla.heading("Semestre", text = "Semestre", anchor = CENTER)
        tabla.heading("Creditos", text = "Creditos", anchor = CENTER)
        tabla.heading("Estado", text = "Estado", anchor = CENTER)
```

Se define la tabla por medio de un Treeview. Asimismo se define dentro de ella el encabezado por medio de una lista que contendrá el Codigo, Nombre, Pre requisito, Opcionalidad, Semestre, Creditos y Estado del curso.

```
    def ingreso_registro():
        for listas in Cursos.listadoCursos:
            tabla.insert(parent = "", index = "end", values = (listas[0], listas[1], listas[2],
Cursos.get_obligatorio_status(int(listas[3])), listas[4], listas[5], Cursos.set_estado_inf(int(listas[6]))))
```

La función ingreso_registro ingresara los cursos que se vayan almacenando para poder mostrarlos en la tabla por fila. Para ello se debe de realizar una iteración para validar que se esté ingresando un curso a la vez. Luego de esto se ingresa cada elemento en una diferente columna.

Clase registrocursos

```
from tkinter import *
from tkinter import messagebox
from cursos import Cursos

class registrocursos(Tk):
    def __init__(self):
        super().__init__()
        self.title("Agregar Cursos")
        self.geometry("400x500")
        self.config(bg = "skyblue")
        self.resizable(False, False)
```

Se crea la clase y se inicializa su constructor con las cualidades de la ventana.

```
        LCodigo = Label(self, text="Codigo")
        LCodigo.place(x = 50, y = 20)
        Codigo = Entry(self, width = "30")
        Codigo.place(x = 180, y = 20)
        LNombre = Label(self, text = "Nombre")
        LNombre.place(x = 50, y = 80)
        Nombre = Entry(self, width = "30")
        Nombre.place(x = 180, y = 80)
        Lpre = Label(self, text = "Pre Requisito")
        Lpre.place(x = 50, y = 140)
        Prerequisito = Entry(self, width = "30")
        Prerequisito.place(x = 180, y = 140)
        LSemestre = Label(self, text = "Semestre")
        LSemestre.place(x = 50, y = 200)
        Semestre = Entry(self, width ="30")
        Semestre.place(x = 180, y = 200)
        LOpcion = Label(self, text = "Opcionalidad")
        LOpcion.place(x = 50, y = 260)
        Opcion = Entry(self, width = "30")
        Opcion.place(x = 180, y = 260)
        LCreditos = Label(self, text = "Creditos")
        LCreditos.place(x = 50, y = 320)
        Creditos = Entry(self, width = "30")
        Creditos.place(x = 180, y = 320)
        LEstado = Label(self, text = "Estado")
        LEstado.place(x = 50, y = 380)
        Estado = Entry(self, width = "30")
        Estado.place(x = 180, y = 380)
```

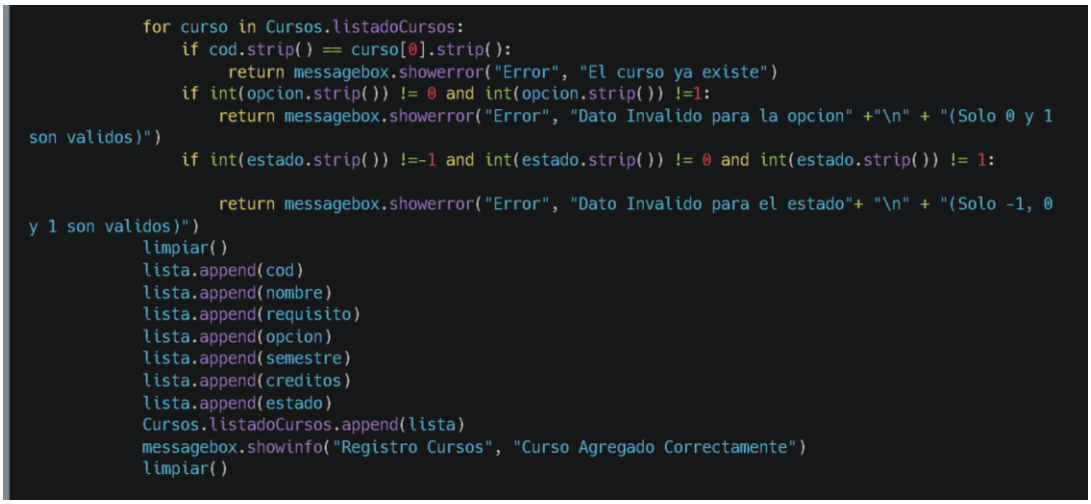
Se crean las herramientas necesarias para definir el formulario en el que el usuario ingresará manualmente los cursos que desea. Para ello debe de utilizarse el elemento Entry que permite entradas de teclado y leer la información ingresada para luego almacenarla en una variable de tipo string.

```
def limpiar():
    Codigo.delete(0, 'end')
    Nombre.delete(0, 'end')
    Prerequisito.delete(0, 'end')
    Opcion.delete(0, 'end')
    Semestre.delete(0, 'end')
    Creditos.delete(0, 'end')
    Estado.delete(0, 'end')
```

La función limpiar permitirá eliminar la información almacenada en los Entry. Esto le ahorrará al usuario tener que borrar cada dato ingresado en los campos de texto.

```
    def agregar_curso():
        cod = Codigo.get()
        nombre = Nombre.get()
        requisito = Prerequisito.get()
        opcion = Opcion.get()
        semestre = Semestre.get()
        creditos = Creditos.get()
        estado = Estado.get()
        lista = []
```

La función agregar_curso agregara los cursos que el usuario ingrese. Primero se definen variables que leerán la información contenida en los Entry. Luego se define una lista vacía que almacenará los cursos que se vayan ingresando uno por uno.

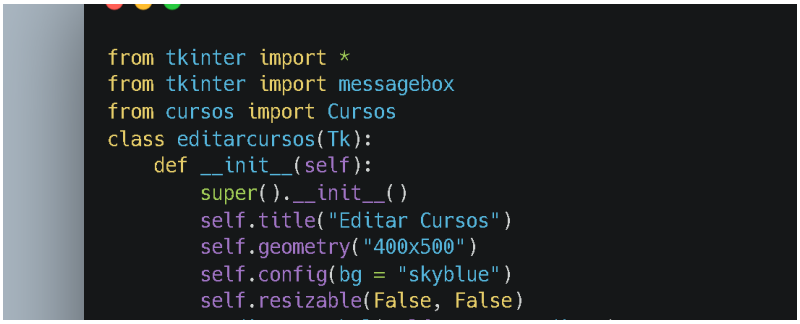


```
            for curso in Cursos.listadoCursos:
                if cod.strip() == curso[0].strip():
                     return messagebox.showerror("Error", "El curso ya existe")
                if int(opcion.strip()) != 0 and int(opcion.strip()) !=1:
                    return messagebox.showerror("Error", "Dato Invalido para la opcion" +"\n" + "(Solo 0 y 1
son validos)")
                if int(estado.strip()) !=-1 and int(estado.strip()) != 0 and int(estado.strip()) != 1:

                    return messagebox.showerror("Error", "Dato Invalido para el estado"+ "\n" + "(Solo -1, 0
y 1 son validos)")
            limpiar()
            lista.append(cod)
            lista.append(nombre)
            lista.append(requisito)
            lista.append(opcion)
            lista.append(semestre)
            lista.append(creditos)
            lista.append(estado)
            Cursos.listadoCursos.append(lista)
            messagebox.showinfo("Registro Cursos", "Curso Agregado Correctamente")
            limpiar()
```

Para agregar los cursos primero se debe de hacer una validación de que no exista el curso dentro de los que han sido cargados. Para ello se debe de verificar que el código que el usuario ingresó no esté dentro de la lista de cursos. Si no lo está la lista vacía definida anteriormente irá cargando elemento por elemento. Si en dado caso ingresa algún dato invalido para la opción de curso o el estado de curso, tirará un mensaje de error especificando cual fue el error y que datos son los únicos validos.

Clase editar_cursos



```
from tkinter import *
from tkinter import messagebox
from cursos import Cursos
class editarcursos(Tk):
    def __init__(self):
        super().__init__()
        self.title("Editar Cursos")
        self.geometry("400x500")
        self.config(bg = "skyblue")
        self.resizable(False, False)
```

Creación de la clase y su constructor. Se definen las cualidades de la ventana.

```
        def editar_curso():
            cod = Codigo.get()
            nombre = Nombre.get()
            requisito = Prerequisito.get()
            opcion = Opcion.get()
            semestre = Semestre.get()
            creditos = Creditos.get()
            estado = Estado.get()
            lista = []
            for curso in Cursos.listadoCursos:
                if cod.strip() == curso[0].strip():
                    Cursos.listadoCursos.remove(curso)
                if int(opcion.strip()) != 0 and int(opcion.strip()) !=1:
                    return messagebox.showerror("Error", "Dato Invalido para la opcion" +"\n" + "(Solo 0 y 1
son validos)")
                if int(estado.strip()) !=-1 and int(estado.strip()) != 0 and int(estado.strip()) != 1:

                    return messagebox.showerror("Error", "Dato Invalido para el estado"+ "\n" + "(Solo -1, 0
y 1 son validos)")
                limpiar()
            lista.append(cod)
            lista.append(nombre)
            lista.append(requisito)
            lista.append(opcion)
            lista.append(semestre)
            lista.append(creditos)
            lista.append(estado)
            Cursos.listadoCursos.append(lista)
            messagebox.showinfo("Edicion Cursos", "Curso " + cod + " Editado Correctamente")
            self.focus()
            limpiar()
```

Como se puede apreciar, la estructura para la clase editar_curso es similar a la de registrar_curso teniendo las mismas validaciones que se explicaron anteriormente. La única diferencia es que, si el código del curso que ingresó el usuario si se encuentra en la lista de cursos, se eliminará totalmente el curso de la lista. Esto se hace para que se pueda ingresar nuevamente el curso con la información cambiada por el usuario en los campos de texto de la ventana. Con esto el curso será actualizado y aparecerá hasta el final de la lista de cursos.

```
        def buscar_curso():
            for lista in Cursos.listadoCursos:
                if str(Codigo.get()) in lista[0]:
                    Nombre.insert(0, lista[1])
                    Prerequisito.insert(0, lista[2])
                    Opcion.insert(0, lista[3])
                    Semestre.insert(0, lista[4])
                    Creditos.insert(0, lista[5])
                    Estado.insert(0, lista[6])
```

Para facilitar la edición de un curso específico se crea la función buscar_curso que devolverá el curso con todos sus atributos en base al código que el usuario ingrese en el Entry. Con esto se enviarán todos los datos de los cursos a los Entry para que el usuario manipule los datos y el sistema los actualice.

Clase eliminarcursos

```
from tkinter import *
from tkinter import messagebox
from cursos import Cursos
class eliminarcursos(Tk):
    def __init__(self):
        super().__init__()
        self.title("Eliminar Cursos")
        self.geometry("400x150")
        self.config(bg = "skyblue")
        self.resizable(False, False)
        LCod = Label(self, text = "Codigo de Curso")
        LCod.place(x = 50, y =30)
        txtcod = StringVar()
        Cod = Entry(self, textvariable =txtcod, width = "30")
        Cod.place(x = 180, y = 30)
        def regresar():
            self.destroy()
        def eliminar_curso():
            cod = Cod.get()
            for curso in Cursos.listadoCursos:
                if cod.strip() == curso[0].strip():
                    Cursos.listadoCursos.remove(curso)
            messagebox.showinfo("Elimninacion Cursos", "Curso " + cod + " Eliminado Correctamente")
            limpiar()
        def limpiar():
            Cod.delete(0, 'end')
        btnEliminar = Button(self, text = "Eliminar", command = eliminar_curso)
        btnEliminar.place(x = 200, y = 100)
        btnRegresar = Button(self, text = "Regresar", command = regresar)
        btnRegresar.place( x = 300, y = 100)
```

La función eliminar_curso buscará el código ingresado por el usuario en el Entry para que al encontrar un curso con dicho código lo pueda eliminar por completo por medio del método **remove**. Con esto el curso sería eliminado totalmente del listado de cursos y se tendría que ingresar de nuevo si se quiere mostrar dentro de la tabla.

Clase conteocreditos

```
    def conteo_creditos_aprobados():
        print("Cursos Aprobados:")
        lista_aprobados = []
        suma = 0
        for curso in Cursos.listadoCursos:
         if int(curso[6]) == 0:
              lista_aprobados.append(curso[1])
              suma = suma + int(curso[5])
        print(lista_aprobados)
        print("\n")
        return suma
```

Esta función buscará dentro del listado de cursos todos aquellos que tengan como estado 0 es decir que sean aprobados. Luego con la variable suma que inicialmente tiene un valor de 0 se irán sumando los créditos para devolver al final la suma total.

```
def creditos_obligatorios():
    print("Cursos hasta el semestre " + Semestre.get())
    suma = 0
    listado = []
    for curso in Cursos.listadoCursos:
       if int(curso[4]) <= int(Semestre.get()):
           if int(curso[3]) == 1:
            listado.append(curso[1])
            suma = suma + int(curso[5])
    print(listado)
    print("\n")
    Creditos_Obligatorios.insert(0, str(suma))
```

La función créditos_obligatorios buscará todos los cursos del listado general que sean de un semestre menor o igual al semestre que indique el usuario en la ventana. Luego verificará si la opción de curso es igual a 1 ósea si el curso es obligatorio. Finalmente sumará los créditos de cada curso y retornará en un Entry la suma total.

Glosario

- Programación Orientada a Objetos (POO): Paradigma de programación que parte del concepto de "objetos" como base, los cuales contienen información en forma de campos y código en forma de métodos.

- Listas: Estructura dinámica de datos que contiene una colección de elementos homogéneos (del mismo tipo) de manera que se establece entre ellos un orden.

- Tkinter: Binding de la biblioteca gráfica Tcl/Tk para el lenguaje de programación Python. Se considera un estándar para la interfaz gráfica de usuario para Python y es el que viene por defecto con la instalación para Microsoft Windows.

- Clases: Agrupación de datos (variables o campos) y de funciones (métodos) que operan sobre esos datos.

- Función: Sección de un programa que calcula un valor de manera independiente al resto de un programa.

- Métodos: Bloque de código que tiene definido en su interior un conjunto de instrucciones las cuales realizan una determinada tarea.