Software para control de pensum

Manual Técnico

Contenido

Backend

En el archivo **Cursos.py** se encuentra el código para cargar el archivo de entrada, gestionar los cursos y el conteo de los créditos.

Cargar Archivos de Entrada

Primero se crea una lista para agregar los cursos que se encuentran en el archivo de entrada. La lista se declara vacía y con el nombre **cursos.**

Se utiliza una función con un parámetro de entrada, este permitirá especificar la ubicación del archivo. La función se declara con el nombre **leer** y con el parámetro **ubicacion**. Para abrir el archivo se utiliza la función propia de Python **open()**, la cual recibe como primer parámetro la ubicación de archivo y un segundo parámetro para indicar las acciones que se realizaran con el archivo. En este caso se utiliza como segundo atributo **"r"** para indicar que el archivo solo será de lectura.

Para leer cada línea del archivo se utiliza un **for in**. En cada ciclo se elimina el salto de línea y se utiliza la función propia de Python **split()** para convertir la cadena de texto en una lista conformada por los elementos del curso.

```
cursos=[]     #Lista de Cursos


#Leer el archivo y Crea la lista de Cursos

def leer(ubicacion):                        #Funcion leer que recibe como parámetro la ubicación del archivo
    archivo = open(ubicacion,"r")           #Abre el archivo
    for linea in archivo:                   #Lee línea a línea el archivo de cursos
        linea = linea.replace("\n","")      #Quita el salto de línea
        if linea=="":                       #Verifica que la línea no se encuentra vacía
            continue                        #Si la línea esta vacía pasa al siguiente ciclo
        linea = linea.split(",")            #Convierte los parámetros de un curso en una lista
```

Antes de añadir el curso a la lista se verifica si este se encuentra repetido dentro de la lista de cursos. Se recorre la lista de cursos con un **for in** para comparar cada elemento, si el curso se repite, se elimina el curso de la lista. La función para verificar se declara con el nombre **verificar** y con un parámetro **linea**. A través de este parámetro se hace saber a la función con quien comparar los cursos de la lista.

```
#Verifica si se repite un curso

def verificar(linea):
    if len(cursos)!=0:                          #Pregunta si la lista se encuentra vacía
        for curso in cursos:                    #Toma un curso de la lista de cursos
            if curso[0]==linea[0]:              #Verifica si se repite
                cursos.remove(curso)            #Si se repite elimina el curso de la lista
```

Después de verificar se añade el curso a la lista de cursos. Al terminar de leer todas las líneas del archivo, este se cierra.

```python
#Leer el archivo y Crea la lista de Cursos

def leer(ubicacion):                        #Funcion leer que recibe como parámetro la ubicación del archivo
    archivo = open(ubicacion,"r")           #Abre el archivo
    for linea in archivo:                   #Lee línea a línea el archivo de cursos
        linea = linea.replace("\n","")      #Quita el salto de línea
        if linea=="":                       #Verifica que la línea no se encuentra vacía
            continue                        #Si la línea esta vacía pasa al siguiente ciclo
        linea = linea.split(",")            #Convierte los parámetros de un curso en una lista
        verificar(linea)
        cursos.append(linea)                #Agrega el curso
    archivo.close                           #Cierra el archivo
```

Gestionar cursos

Listar Cursos

Los cursos agregados se encuentran en la lista **cursos**. En la sección de Frontend se explica como mostrar la lista de cursos.

```python
cursos=[]     #Lista de Cursos
```

Mostrar Curso

Se declara una función con el nombre **buscar** y con un parámetro llamada **codigo**. A través del parámetro se indica el código del curso que se debe buscar en la lista **cursos**. Si el curso existe en la lista **cursos**, la función devuelve el curso en formato de lista. De lo contrario devuelve **None**, indicando que el curso no existe.

```python
#Busca Curso por Código

def buscar(codigo):
    if len(cursos)!=0:                      #Verifica si la lista de cursos esta vacía
        for curso in cursos:                #Recorre la lista de cursos
            if curso[0]==str(codigo):       #Verifica el Código del curso
                return curso                #Si el curso coincide lo devuelve
    return None                             #Si el curso no existe devuelve None
```

Agregar Curso

Se declara una función con el nombre **agregar** y con un parámetro llamado **curso**. A través del parámetro se indica el curso que se desea agregar a la lista **cursos**. El curso debe estar el formato de lista, donde cada posición de la lista es un elemento del curso.

Formato

```python
["Código","Nombre","Prerrequisitos","Obligatorio","Semestre","Créditos","Estado"]
```

Haciendo uso de la función **verificar** anteriormente declarada, se verifica si el curso a agregar existe en la lista **cursos**. Si ya existe un curso con el mismo código, se elimina y se agregar el nuevo curso que se proporciono por el parámetro de la función en el formato indicado.

```
#Agregar Curso

def agregar(curso):
    verificar(curso)        #Verifica si el curso existe
    cursos.append(curso)
```

Editar curso

Se utilizan las funciones **buscar** y **agregar** anteriormente declaradas. Su implementación se explica en la sección de Frontend.

Eliminar curso

Se declara una función con el nombre **eliminar** y con un parámetro llamado **codigo**. Con el método **remove** propio de una lista de Python, se elimina el curso que se indica en su argumento. Para indicar el curso que se desea eliminar se utilizara la función **buscar** anteriormente declarada, ya que retorna un curso con el código que se ingresa como parámetro. Todo el proceso esta dentro de un **try except** debido a que, si el curso que se desea eliminar de la lista de cursos no existe, el método **remove** genera un error de tipo **ValueError**.

```
#Eliminar un curso por Código

def eliminar(codigo):
    try:
        cursos.remove(buscar(codigo))     #Busca y elimina el curso con el Código
        return True                       #Retorna True si fue eliminado con éxito
    except ValueError:
        return False                      #Retorna False indicando que el curso no existe
```

Conteo de créditos

Créditos aprobados

Se declara una función con el nombre **aprobados** y una variable llamada **creditos** inicializada en cero que contendrá la suma de los créditos totales. Con un **for in** se recorre la lista **cursos** y por cada ciclo se compara la posición 6 del curso, ya que esta describe el Estado del curso. Si el Estado del curso es 0 (equivale a aprobado), entonces los créditos del curso se suman a la variable **creditos**. Al terminar de recorrer la lista **cursos**, la función retornar la suma de créditos totales. Si la lista cursos se encuentra vacia, la función solamente retorna la variable **creditos** que se encuentra inicializada en cero.

```
#Creditos aprobados

def aprobados():
    creditos=0
    if len(cursos)!=0:                  #Verifica si la lista esta vacía
        for curso in cursos:            #Recorre la lista cursos
            if int(curso[6])==0:        #Verifica que el curso este aprobado
                creditos+=int(curso[5]) #Si esta aprobado suma los créditos
        return creditos                 #Retorna los créditos totales
    return creditos                     #Retorna créditos que vale cero
```

Créditos cursando

Se declara la función con el nombre **cursando**. La función utiliza la misma estructura que la función **aprobados** anteriormente declarada. El único cambio es que los créditos del curso se sumaran si el Estado es 1 (equivale a cursando).

```
#Creditos Cursando

def cursando():
    creditos=0
    if len(cursos)!=0:                  #Verifica si la lista esta vacía
        for curso in cursos:            #Recorre la lista cursos
            if int(curso[6])==1:        #Verifica que el curso este en estado cursando
                creditos+=int(curso[5]) #Si se esta cursando suma los créditos
        return creditos                 #Retorna los créditos totales
    return creditos                     #Retorna créditos que vale cero
```

Créditos pendientes

Se declara la función con el nombre **pendientes**. La función utiliza la misma estructura que la función **aprobados** anteriormente declarada. El único cambio es que ahora los créditos se sumaran si el Estado del curso sea -1 (equivale a pendientes) y también que la posición 3 del curso que describe si el curso es obligatorio sea 1(equivale a obligatorio).

```
#Creditos Pendientes

def pendientes():
    creditos=0
    if len(cursos)!=0:                              #Verifica si la lista esta vacía
        for curso in cursos:                        #Recorre la lista cursos
            if int(curso[3])==1 and int(curso[6])==-1:  #Verifica que el curso este en estado pendiente y que sea obligatorio
                creditos+=int(curso[5])             #Suma los créditos
        return creditos                             #Retorna los créditos totales
    return creditos                                 #Retorna créditos que vale cero
```

Créditos hasta semestre N

Se declara la función con el nombre **hasta_N** con el parámetro llamado **N**. El parámetro **N** indica el numero de semestre hasta donde se sumarán los créditos de los cursos. Para que los créditos se sumen, la posición 3 del curso tendrá que ser 1 (indica que es obligatorio) y la posición 4 (indica el semestre del curso) tendrá que ser menor o igual a **N**.

```
#Creditos hasta semestre N

def hasta_N(N):
    creditos=0
    if len(cursos)!=0:                              #Verifica si la lista esta vacía
        for curso in cursos:                        #Recorre la lista cursos
            if int(curso[3])==1 and int(curso[4])<=N:  #Verifica que el curso sea obligatorio y el semestre menor o igual a N
                creditos+=int(curso[5])             #Suma los créditos
        return creditos                             #Retorna los créditos totales
    return creditos                                 #Retorna créditos que vale cero
```

Créditos del semestre

Se declara la función con el nombre **semestre** y el parámetro llamado **N**. El parámetro **N** indica el numero de semestre a los que los cursos tienen que pertenecer para sumar los créditos. También se declaran 3 variables llamadas **aprobadas**, **asignados** y **pendientes** inicializadas en cero. Se recorre la lista **cursos** y por cada ciclo se verifica que el curso pertenezca al semestre indicado y se utiliza un **case** para comprobar el Estado del curso y sumar los créditos a la variable correspondiente al Estado.

```python
#Creditos del Semestre

def semestre(N):
    aprobados=0
    asignados=0
    pendientes=0

    if len(cursos)==0:                          #Verifica si la lista esta vacía
        return [0,0,0]                          #Retorna una lista que indica cero creditos

    for curso in cursos:                        #Recorre la lista cursos
        if int(curso[4])!=N:                    #Verifica que el curso pertenezca al semestre
            continue                            #Si no pertenece pasa al siguiente curso
        match int(curso[6]):                    #Verifica eel Estado del curso
            case 0 :
                aprobados+=int(curso[5])        #Suma los créditos si esta aprobado
            case 1 :
                asignados+=int(curso[5])        #Suma los créditos si esta cursando
            case -1 :
                pendientes+=int(curso[5])       #Suma los créditos si esta pendiente
    return [aprobados,asignados,pendientes]     #Retorna la suma total en una lista.
```

Frontend

Para crear las ventanas con las que el usuario interactuara se utiliza la librería **tkinter**. Todas las configuraciones de las ventanas se encuentran en el archivo **interfaz.py**.

Para poder utilizar las funciones y la lista de cursos creadas anteriormente, se importarán al archivo **interfaz.py** de la siguiente forma:

```python
from Cursos import *
```

Creación y configuración de Ventanas

Las diferentes ventanas que se utilizan en la aplicación serán declaradas en funciones dentro de una sola clase llamada **interfaz**. Cada función representa una ventana de la aplicación y en cada una se configura su Widget.

En el constructor de la clase **interfaz** se instanciará la ventana y Widgets de la ventana principal denominada **menu**. **Tk()** es propio de la librería **tkinter**.

```python
class interfaz():
    def __init__(self):
        #conf Ventana
        self.menu = Tk()
```

Después se instancia un **Frame** con el nombre **marco**, en este **Frame** es donde se colocarán todos los Widgets. Los Widgets que se utilizan para la configuración de todas las ventas son **Button**, **Label** y **TreeView**.

```
class interfaz():
    def __init__(self):
        #conf Ventana
        self.menu = Tk()
        self.menu.resizable(False,False)
        self.menu.title("Menu")
        self.menu.geometry("500x500+"+str((self.menu.winfo_screenwidth()-500)//2)+"+"+str((self.menu.winfo_screenheight()-500)//2))
        self.menu.attributes("-topmost",True)
        #conf Frame
        self.marco=Frame(self.menu,borderwidth=10,relief="sunken")
        self.marco.place(x=20,y=20,width=460,height=460)
        #conf componentes del Frame
        self.info = Label(self.marco,text="Lenguajes Formales Y De Programacion\n\nNombre : Sergio Saul Ralda Mejia\n\nCarne : 202103216")
        self.info.place(x=95,y=20,width=250,height=100)
        self.bcargar=Button(self.marco,text="Cargar Archivo",command=self.ir_carga_archivo)
        self.bcargar.place(x=170,y=150,width=100,height=25)
        self.bgestionar=Button(self.marco,text="Gestionar Cursos",command=self.ir_gestion_cursos)
        self.bgestionar.place(x=170,y=195,width=100,height=25)
        self.conteo=Button(self.marco,text="Conteo de Creditos",command=self.ir_conteo_creditos)
        self.conteo.place(x=160,y=240,width=120,height=25)
        self.salir=Button(self.marco,text="Salir",command=self.menu.destroy)
        self.salir.place(x=170,y=285,width=100,height=25)
        #Inicia la ventana Menu
        self.menu.mainloop()
```

Para cada ventana que se utiliza en la aplicación, se crea una función y en ella se utiliza la misma estructura que en el constructor de la clase **interfaz**. El siguiente Código es la configuración de todos los Widgets para la ventana Agregar Curso.

```
#Ventana para Agegar curso
def agregar_curso(self):
    #conf venta
    self.agregar = Tk()
    self.agregar.resizable(False,False)
    self.agregar.title("Agregar Curso")
    self.agregar.geometry("500x500+"+str((self.agregar.winfo_screenwidth()-500)//2)+"+"+str((self.agregar.winfo_screenheight()-500)//2))
    self.agregar.attributes("-topmost",True)
    #conf Frame
    self.marco6=Frame(self.agregar,border=10, relief="sunken")
    self.marco6.place(x=20,y=20,width=460,height=460)
    #conf elementos
    self.lcodigo=Label(self.marco6,text="Codigo")
    self.lcodigo.place(x=55,y=30,width=100,height=30)
    self.Ecodigo=Entry(self.marco6)
    self.Ecodigo.place(x=185,y=30,width=200,height=30)
    self.lnombre=Label(self.marco6,text="Nombre")
    self.lnombre.place(x=55,y=80,width=100,height=30)
    self.Enombre=Entry(self.marco6)
    self.Enombre.place(x=185,y=80,width=200,height=30)
    self.lprerequisito=Label(self.marco6,text="Pre Requisito")
    self.lprerequisito.place(x=55,y=130,width=100,height=30)
    self.Eprerequisito=Entry(self.marco6)
    self.Eprerequisito.place(x=185,y=130,width=200,height=30)
    self.lobligatorio=Label(self.marco6,text="Obligatorio")
    self.lobligatorio.place(x=55,y=180,width=100,height=30)
    self.Eobligatorio=Entry(self.marco6)
    self.Eobligatorio.place(x=185,y=180,width=200,height=30)
    self.lsemestre=Label(self.marco6,text="Semestre")
    self.lsemestre.place(x=55,y=230,width=100,height=30)
    self.Esemestre=Entry(self.marco6)
    self.Esemestre.place(x=185,y=230,width=200,height=30)
    self.lcreditos=Label(self.marco6,text="Creditos")
    self.lcreditos.place(x=55,y=280,width=100,height=30)
    self.Ecreditos=Entry(self.marco6)
    self.Ecreditos.place(x=185,y=280,width=200,height=30)
    self.lestado=Label(self.marco6,text="Estado")
    self.lestado.place(x=55,y=330,width=100,height=30)
    self.Eestado=Entry(self.marco6)
    self.Eestado.place(x=185,y=330,width=200,height=30)
    self.bagregar_curso=Button(self.marco6,text="Agregar",command=self.insertar_curso)
    self.bagregar_curso.place(x=170,y=380,width=100,height=30)
    self.bregresar_agregar=Button(self.marco6,text="Regresar",command=self.regresar_gestion_agregar)
    self.bregresar_agregar.place(x=0,y=410,width=100,height=30)
    #iniciar ventana agregar curso
    self.agregar.mainloop()
```

Cargar Archivo de Entrada

Se declara una función dentro de la clase **interfaz** llamada **cargar_archivo**. Se utiliza la función **leer** anteriormente declarada y como argumento la ruta del archivo. La función **leer** esta dentro de un **try except** debido a que, si la ruta del archivo no existe generara un error de tipo **FileNotFoundError**.

```
    #Proceso para leer y almacenar los cursos del archivo
def cargar_archivo(self,ruta):
    try:
        leer(ruta)
        showinfo(title="Mensaje",message="Archivo Cargado Correctamente")
        self.regresar_menu()
    except FileNotFoundError:
        showerror(title="Error",message="El Archivo NO Existe")
```

La función **cargar_archivo** se llamará en el botón creado en la ventana **cargar**.

```
self.baceptar=Button(self.marco2,text="Cargar Archivo",command= lambda: self.cargar_archivo(self.truta.get()))
```

Gestionar Cursos

Listar cursos

Dentro de la función donde se crea la ventana para ver la lista de cursos se utiliza un **for in** para recorrer la lista **cursos** anteriormente declarada. Se instancia una tabla con el **Widgets Treeview** para mostrar los cursos. En cada ciclo del **for in** se añade un curso a la tabla.

```
for curso in cursos:
    self.obligatorio=lambda a=str(curso[3]): "Si" if (a=="1") else "No"
    self.estado=lambda a=str(curso[6]): "Aprobado" if (a=="0") else ("Cursando" if (a=="1") else "Pendiente")
    self.prerrequisito=lambda a=str(curso[2]): str(curso[2]) if(a!="") else "Ninguno"
    self.tabla.insert("",END,text=str(curso[0]),values=(str(curso[1]),self.prerrequisito(),self.obligatorio(),str(curso[4]),str(curso[5]),self.estado()))
```

Mostrar curso

Se utiliza la función **buscar** anteriormente declarada y como argumento el código del curso que se quiere mostrar. La función retornara una lista con todos los elementos del curso para poder insertarlo en una tabla creada utilizando el **Widgets Treeview**.

```
self.curso=buscar(self.codigo.get())
if self.curso!=None:
    self.obligatorio=lambda a=str(self.curso[3]): "Si" if (a=="1") else "No"
    self.estado=lambda a=str(self.curso[6]): "Aprobado" if (a=="0") else ("Cursando" if (a=="1") else "Pendiente")
    self.prerrequisito=lambda a=str(self.curso[2]): str(self.curso[2]) if(a!="") else "Ninguno"
    self.tabla2.insert("",END,text=str(self.curso[0]),values=(str(self.curso[1]),self.prerrequisito(),self.obligatorio(),str(self.curso[4]),str(self.curso[5]),self.estado()))
else:
    showerror(title="Error",message="Curso No Ecntrado")
```

Agregar curso

Se declara una función con el nombre **insertar_curso** que se encarga de recolectar todos los datos proporcionados por el usuario para crear el nuevo curso. Con los datos para el nuevo curso se forma una lista. Se utiliza la función **agregar** anteriormente declara y como argumento se coloca la lista formada con los datos.

```
def insertar_curso(self):
    curso=[self.Ecodigo.get(),self.Enombre.get(),self.Eprerequisito.get(),self.Eobligatorio.get(),self.Esemestre.get(),self.Ecreditos.get(),self.Eestado.get()]
    if curso[0]=="":
        showwarning(title="Advertencia",message="Ingrese Un Codigo Como Minimo")
        return None
    agregar(curso)
```

Esta función se manda llamar en el botón creado en la función para la ventana de Agregar Curso.

```
self.bagregar_curso=Button(self.marco6,text="Agregar",command=self.insertar_curso)
```

Editar curso

Se declara una función con el nombre **editar_elcurso**. Se utiliza las funciones **buscar** y **agregar** anteriormente declaradas. Con la función **buscar** se verifica que el código del curso que se quiere editar exista. Si el curso existe la función **buscar** retornara un curso en formato de lista. Después se utiliza la función **agregar** para eliminar el curso existente y agregar el nuevo curso. A la función **agregar** se le indica como argumento la lista del curso retornada por la función **buscar**.

```
    #Buscar el curso y lo edita
def editar_elcurso(self):
    codigo=self.Ecodigo.get()
    if codigo!="":
        curso=buscar(codigo)
        if curso!=None:
            curso=[codigo,self.Enombre.get(),self.Eprerequisito.get(),self.Eobligatorio.get(),self.Esemestre.get(),self.Ecreditos.get(),self.Eestado.get()]
            agregar(curso)
```

Eliminar curso

Se declara una función con el nombre **eliminar_elcurso**. Se utiliza función **eliminar** anteriormente declarada y se le indica como argumento el código proporcionado por el usuario.

```
    #Elimina el curso con el codigo ingresado
def eliminar_elcurso(self):
    codigo=self.Ecodigo.get()
    if codigo!="":
        if eliminar(codigo):
            self.Ecodigo.delete(0,END)
            showinfo(title="Mensaje",message="Curso Eliminado con Exito")
        else:
            showerror(title="Error",message="El Curso no Existe")
    else:
        showwarning(title="Advertencia",message="Ingrese un Codigo")
```

Conteo de créditos

Se utilizan las funciones **aprobados**, **cursando** y **pendientes** anteriormente declaradas. Estas funciones se mandan a llamar en el **Label** correspondiente en el atributo texto declaradas en la función para crear la ventana Conteo de créditos.

```
self.lcantaprobados=Label(marco,text=str(aprobados()))
```

```
self.lcantcursando=Label(marco,text=str(cursando()))
```

```
self.lcantpendientes=Label(marco,text=str(pendientes()))
```

Se declara una función con el nombre **contar_hastaN** para recibir el numero de semestre proporcionado por el usuario e indicarlo como argumento en la función **hasta_N** anteriormente declarada.

```
def contar_hastN(self):
    try:
        N=int(self.Esemestre.get())
        if N<1:
            int("")
    except ValueError:
        self.lcanthastaN.config(text="---")
        showerror(title="Error",message="Semestre Invalido")
        return None
    creditos=str(hasta_N(N))
    self.lcanthastaN.config(text=creditos)
    showinfo(title="Mensaje",message="Creditos Contados con Exito")
```

Se declara una función mas con el nombre **contar_N** para recibir el numero de semestre proporcionado por el usuario e indicarlo como argumento en la función **semestre** anteriormente declarada.

```
def contar_N(self):
    try:
        N=int(self.Esemestre2.get())
        if N<1:
            int("")
    except ValueError:
        self.lcantN.config(text="---")
        showerror(title="Error",message="Semestre Invalido")
        return None
    creditos=semestre(N)
    creditos="#Aprobados: "+str(creditos[0])+"   #Asignados: "+str(creditos[1])+"   #Pendientes: "+str(creditos[2])
    self.lcantN.config(text=creditos)
    showinfo(title="Mensaje",message="Creditos Contados con Exito")
```