Manual técnico 2023

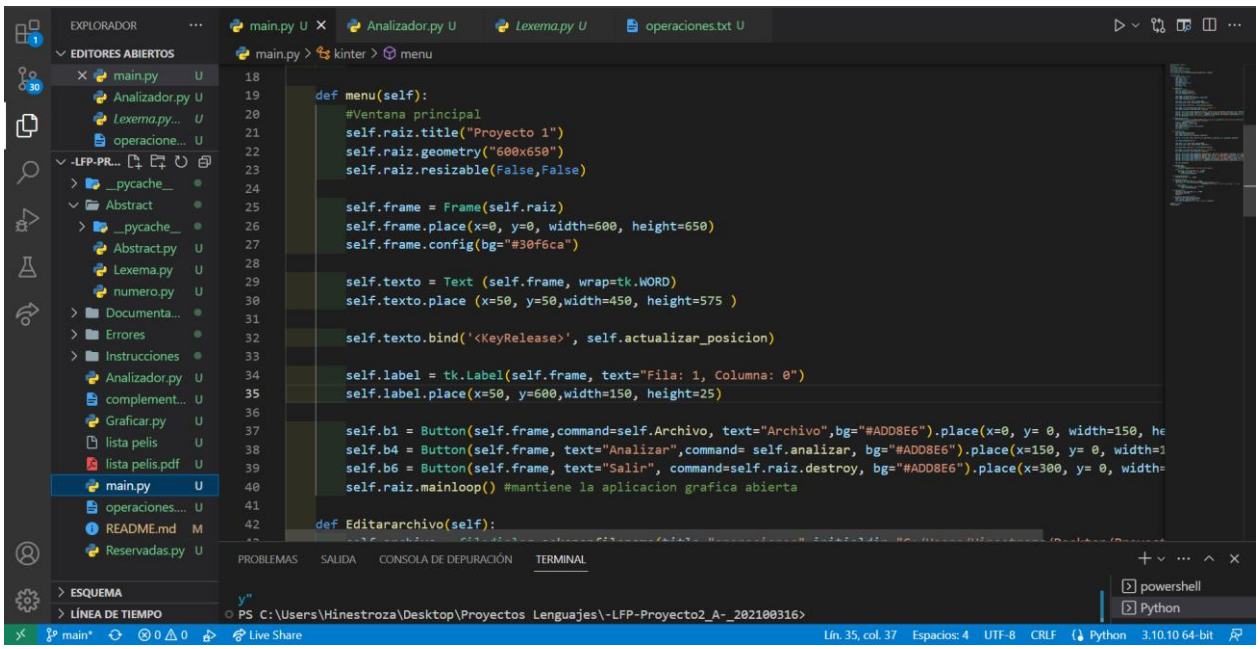
25 ABRIL

Universidad San Carlos de Guatemala
Creado por: Jose Andres Hinestroz García

Nombre del logotipo

Título de encabezado

La siguiente práctica se diseñó a base de lenguaje de Python usando Visual Code como IDE. Se implementó el lenguaje de programa de orientación de objetos y además clases y métodos abstractos que nos ayudarán a heredar a otras clases sus funcionalidades. Se realizaron varias clases para poder distinguir entre cada una en el cual el usuario se podrá interactuar, identificándose según funcionalidad, además de tener una interfaz gráfica, que nos ayudará a navegar a través de lo que queramos



Nuestra clase main nos implementara nuestra parte gráfica del proyecto con la cuales en su menu podemos ver los diferentes botones que cada uno de este posee, como demas con los cuales cada unno de estos botone, posee un metodo con el cual nos ayudara a implementar la funcion que nosotros deseamos realizar. Ademas de una usamos bind la cual esta es un método de los widgets en Tkinter que permite vincular un evento específico (en este caso, <KeyRelease>) con una función callback (en este caso, actualizar_posicion). Cuando ocurre el evento, se llama automáticamente. Asimismo <KeyRelease>: es el evento que se activa cuando se suelta una tecla. En este caso, queremos que la función actualizar_posicion se ejecute cada vez que se libere una tecla en el widget de texto.

El siguiente método nos va ayudar para poder llamar el archivo que nosotros queramos abrir.

```
def Editararchivo(self):
    self.archivo = filedialog.askopenfilename(title="operaciones",initialdir="C:/Users/Hinestroza/Desktop/Proy
                   filetypes=[("Archivo txt", "*.txt")])
    archivo = open(self.archivo,"r")
    self.text_content = archivo.read()
    self.eliminartexto()
    self.texto.insert(tk.END,self.text_content)
    self.bandera = False
    #print(self.archivo)
```

El siguiente metodo tiene una estructura parecida a la del menu ya que estas contienen las mismas funciones, sin embargo, en el menu trabaja con lo que nosotros ingresamos en el cuadro de texto, mientars que el siguiente es para llamar a un archivo y que este lo cargue y podramos editarlo.

```
def Vertexto(self, direccion):
    self.raiz.geometry("1000x575")
    self.frame = Frame(self.raiz)
    self.b2 = Button(self.frame, text="Guardar", bg="#ADD8E6").place(x=700, y= 60, width=150, height=30)
    self.b3 = Button(self.frame, text="Guardar Como",bg="#ADD8E6").place(x=700, y= 90, width=150, height=30)
    self.frame.place(x=0, y=0, width=1000, height=575)
    self.frame.config(bg="#30f6ca")
    self.texto = Text (self.frame)
    self.texto.place (x=50, y=5,width=550, height=550 )
    archivo = open(direccion,"r")
    text_content = archivo.read()
    self.texto.insert(tk.END,text_content)
    archivo = open(direccion,"r")

    lineas = ""
    for i in archivo.readlines():
        lineas +=i
    instruccion(lineas)
    _operar()
    self.raiz.mainloop()
```

Para la creacion de nuestro arbol se tuvo que hacer varios if, con el objetivo de que si era simbolo, corchete operacione, valor o un numero que estos fueran a operarse de diferetne manera, con las cuales si era un nos ayudarian dos metodos diferentes las cuales una armaba un lexema o bien un numero con el cual vamos a operar.

Como ademas tenemos diferentes metodos que nos ayudaran a los botones a funcionar, los cuales tenemos de limpiar el cuadro, gurdar el archivo que selccionamos o bien guardar un archivo nuevo.

```
def guardar(self):
    if self.bandera:
        messagebox.showinfo("INFO", "No hay archivo cargado")
    else:
        contenido = self.texto.get('1.0', tk.END)
        with open(self.archivo, 'w') as archivo:
            archivo.write(contenido)

def eliminartexto(self):
    self.texto.delete('1.0', tk.END)

def guardar_como(self):
    contenido = self.texto.get('1.0', tk.END)
    ruta_archivo = filedialog.asksaveasfilename(defaultextension='.txt',
                                                filetypes=[('Text Files', '*.txt'), ('All Files', '*.*')])
    if ruta_archivo:
        with open(ruta_archivo, 'w') as archivo:
            archivo.write(contenido)
```

Nuestra clase abstratcta con el cual heredan sus atributos otras clases como seria el lexema y el numero lass cuales estas al haber separado los lexemas estos los mandamos a sus

```
from abc import ABC, abstractmethod

class Expresion(ABC):
    def __init__(self,fila,columna) -> None:
        self.fila = fila
        self.columna = columna

    @abstractmethod #todo se opera
    def operar(self,arbol):#manejamos arbol
        pass

    @abstractmethod
    def getfila(self):
        return self.fila
    @abstractmethod
    def getcolumna(self):
        return self.columna
```

```python
from Abstract.Abstract import Expresion
from math import *
class Arimetica(Expresion):
    def __init__(self, izquierda,derecha, tipo, fila,columna) -> None:
        self.izquierda = izquierda
        self.derecha = derecha
        self.tipo = tipo
        super().__init__(fila, columna)

    def operar(self, arbol):
        leftvalue = ''
        rightvalue = ''
        #verificar que los numeros no vengan vacios
        if self.izquierda != None:
            leftvalue = self.izquierda.operar(arbol)
        if self.derecha != None:
            rightvalue = self.derecha.operar(arbol)

        if self.tipo.operar(arbol) == 'CrearBD':
            return 'use(‘nombreBaseDatos’);'
        elif self.tipo.operar(arbol) == 'EliminarBD':
            return 'db.dropDatabase();'
        elif self.tipo.operar(arbol) == 'CrearColeccion':
            return 'db.createCollection(‘nombreColeccion’); '
        elif self.tipo.operar(arbol) == 'EliminarColeccio':
            return 'db.nombreColeccion.drop();'
```