MANUAL TECNICO

PROYECTO #1 LFP B+

Diego Facundo Pérez Nicolau
CARNET 202106538

Python:

Python es un lenguaje de alto nivel de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código. Se trata de un lenguaje de programación multiparadigma, ya que soporta parcialmente la orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma.

Programación Orientada a Objetos:

La programación orientada a objetos (POO) es una manera de estructurar el código que le hace especialmente efectivo organizando y reutilizando código, aunque su naturaleza abstracta hace que no sea muy intuitivo cuando se empieza. En general, los objetos se pueden considerar como tipos de datos con características propias que también pueden tener funcionalidades propias. De forma similar podemos crear nuestros propios objetos con características (llamadas atributos) y métodos propios. Los objetos se definen usando clases (class()) y las variables que de definen en ella son propiedades comunes de ese objeto.

Listas:

En Python, para abrir un archivo usamos la función open, que recibe el nombre del archivo a abrir. Esta función intentará abrir el archivo con el nombre indicado. Si tiene éxito, devolverá una variable que nos permitirá manipular el archivo de diversas maneras. El contenido de la ventana de Python se puede guardar en un archivo de Python o un archivo de texto. Haga clic con el botón derecho del ratón en la ventana de Python y seleccione Guardar como para guardar el código como un archivo de Python (. py) o un archivo de texto.

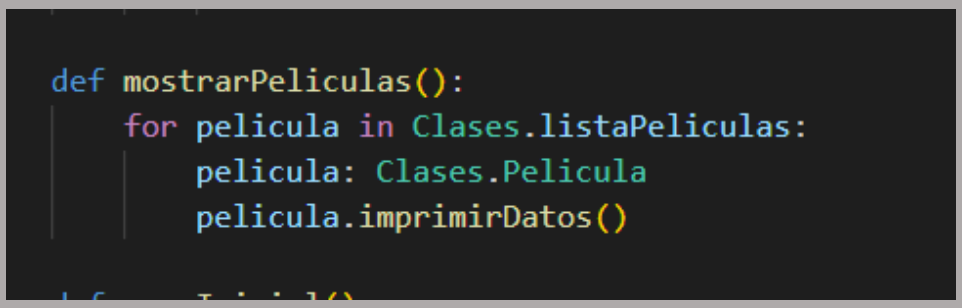
Tkinter:

Tkinter es un binding de la biblioteca gráfica Tcl/Tk para el lenguaje de programación Python. Se considera un estándar para la interfaz gráfica de usuario (GUI) para Python y es el que viene por defecto con la instalación para Microsoft Windows. El paquete tkinter es la interfaz por defecto de Python para el kit de herramientas de GUI Tk. Viene integrado con Python y no es necesario instalarlo. Es muy buena opción para quienes van empezando.

Declaración de variables métodos y clases

Variables Y Métodos:

Las variables y métodos se declararon emperezando y manteniéndolas en minúsculas (excepto si eran acrónimos) y si incluían más de una palabra para separar una palabra de otra la nueva palabra se mantenía pegada pero su primera letra era mayúscula:

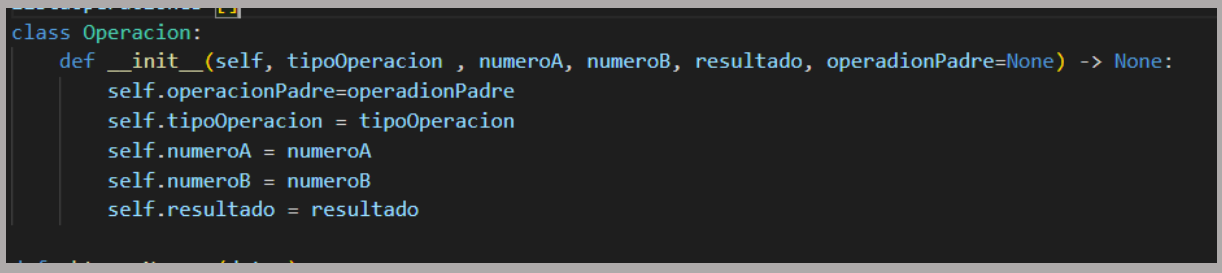


```
def mostrarPeliculas():
    for pelicula in Clases.listaPeliculas:
        pelicula: Clases.Pelicula
        pelicula.imprimirDatos()
```

(ejemplo dónde también se resume una palabra en las variables)

Clases:

Las clases tienen que estar inicializadas con letra mayúscula seguidas de minúsculas hasta terminar la palabra y la siguiente palabra también empezarse por mayúscula.



```
class Operacion:
    def __init__(self, tipoOperacion , numeroA, numeroB, resultado, operadionPadre=None) -> None:
        self.operacionPadre=operadionPadre
        self.tipoOperacion = tipoOperacion
        self.numeroA = numeroA
        self.numeroB = numeroB
        self.resultado = resultado
```

Métodos

1. Menú Principal: Usando la librería de Tkinter se creará una ventana la cual redirige a otros dos menús.

```
import tkinter as tk

def abrirMenuArchivos():
    import MenuArchivo
    MenuArchivo.abrirWindowMA()

def abrirMenuAyuda():
    import InterfazAyuda.MenuAyuda as MenuAyuda
    MenuAyuda.abrirWindowA()

windowInicio = tk.Tk()
windowInicio.title("Inicio") #Asignarle titulo a ala ventana
windowInicio.columnconfigure([0,4], minsize=200) #Columnas de la Ventana
windowInicio.rowconfigure([0,10], minsize=100) #Filas de la Ventana y su proporción
windowInicio.configure(background="#00d2d3")

button1= tk.Button(windowInicio, text ="Abrir Menu de Archivo", command=lambda: abrirMenuArchivos(),  bg="#25CCF7")
button1.grid(row=3,column=1, pady=10)

button2= tk.Button(windowInicio, text ="Ayuda", command=lambda: abrirMenuAyuda(), bg="#55E6C1") #Boton de redirrecio
button2.grid(row=6,column=1, pady=10)

button4= tk.Button(windowInicio, text ="Salir", command=windowInicio.destroy, bg="#ee5253")   #Botón de salir
button4.grid(row=9,column=1)

windowInicio.mainloop()              #Llamamos a la ventana
```

2. Abrir Archivo (lectura) y Guardar Archivo:

```
def revisar(ruta):

        archivo = open (ruta,'r')       #Se abre archivo segun la ruta indicada
        datos = archivo.read()          #Se extrae la informacion de este
        datosErrores = revisarErrores(datos)    #Devuelve si tiene errores con formato
        archivo.close()                 #cerramos ese archivo

        newFile = open ("Errores/ERRORES_202106538", "w")       #Creamos archivo con la ruta quemada para escribir
        newFile.write(datosErrores)                             #Escribimos los datos de los Errores
        newFile.close                                           #Cerramos el archivo
```

3. Revisión de Errores:

```
def revisarErrores(datos):
    info='{'

    nFila=0
    nError=0

    lineas =datos.split("\n")   #Se divide por lineas
    for linea in lineas:
            nFila=nFila+1
            nColumna =0

            for c in linea:     #se revisa caracter por caracter de la linea
                    nColumna=nColumna+1

                    if not(c.isdigit()|c.isalpha() or c==" " or c == "{" or c=="}" or c == ":" or c==","  or c == "[" or c=="]" or c == "=" or c == "."or c == '"' or c == "-"):
                            nError=nError+1

                            print("Error #", nError, ". ",c, "Fila: ",nFila, "  Columna: ",nColumna)
                            if not(nError==1):
                                info+=','

                                info+=formatoError(nError,c,nColumna,nFila)

    if nError ==0:
            print('No hay errores')
    #print(info)
    return info
```

4. Guardado de Operaciones.

```
def EV1(texto):
    return EV(texto,'"Valor1":')
def EV2(texto):
    try:
        return EV(texto,'"Valor2":')
    except:
        return None

def resultadoOperacion(text):

    tipoOp = ETipoOperacion(text)
    #print("Operacion: "+tipoOp)
    valor1 = EV1(text)
    valor2 = EV2(text)
    resultado = FuncionOperaciones.hacer(tipoOp,valor1,valor2)
    print(resultado)

    listaOperaciones.append(Operacion(tipoOp,valor1,valor2,resultado))
    return resultado

def realizarOperaciones(texto:str):
    #print(texto)
    data=texto.split(',')
    for element in data:
        resultadoOperacion(element)
```

5. Creación de grafico:

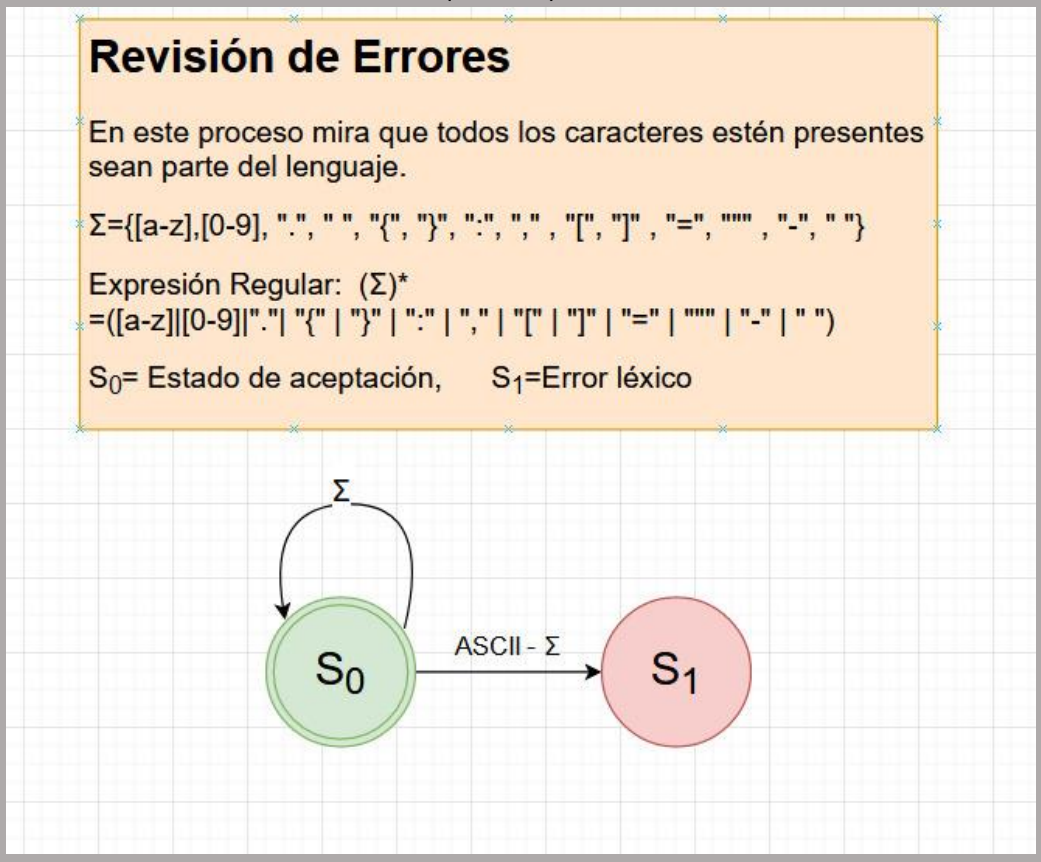
```
def infoGrafica(texto):
    formaN=Formas[EncontrarFormaNodo(texto)]
    colorN=Colores[EncontrarColorNodo(texto)]
    colorF=colorL=Colores[EncontrarColorFuente(texto)]

    textoGrafica='digraph dot\n{\n'
    textoGrafica+='\tnode[shape='+formaN+', fontcolor='+colorL+', fillcolor='+colorN+', style=filled];\n'
    textoGrafica+='\n'

    for operacion in listaOperaciones:
        operacion : Operacion
        textoGrafica+='\t"'+ operacion.tipoOperacion +'\\n '+str(operacion.resultado)+'"\n'


        if operacion.numeroA!=None:
            textoGrafica+='\t"'+ operacion.tipoOperacion +'\\n '+str(operacion.resultado)+'"->'+str(operacion.numeroA)+'\n'

        if operacion.numeroB!=None:
            textoGrafica+='\t"'+ operacion.tipoOperacion +'\\n '+str(operacion.resultado)+'"->'+str(operacion.numeroB)+'\n'

    textoGrafica+='}'
```

Descripción

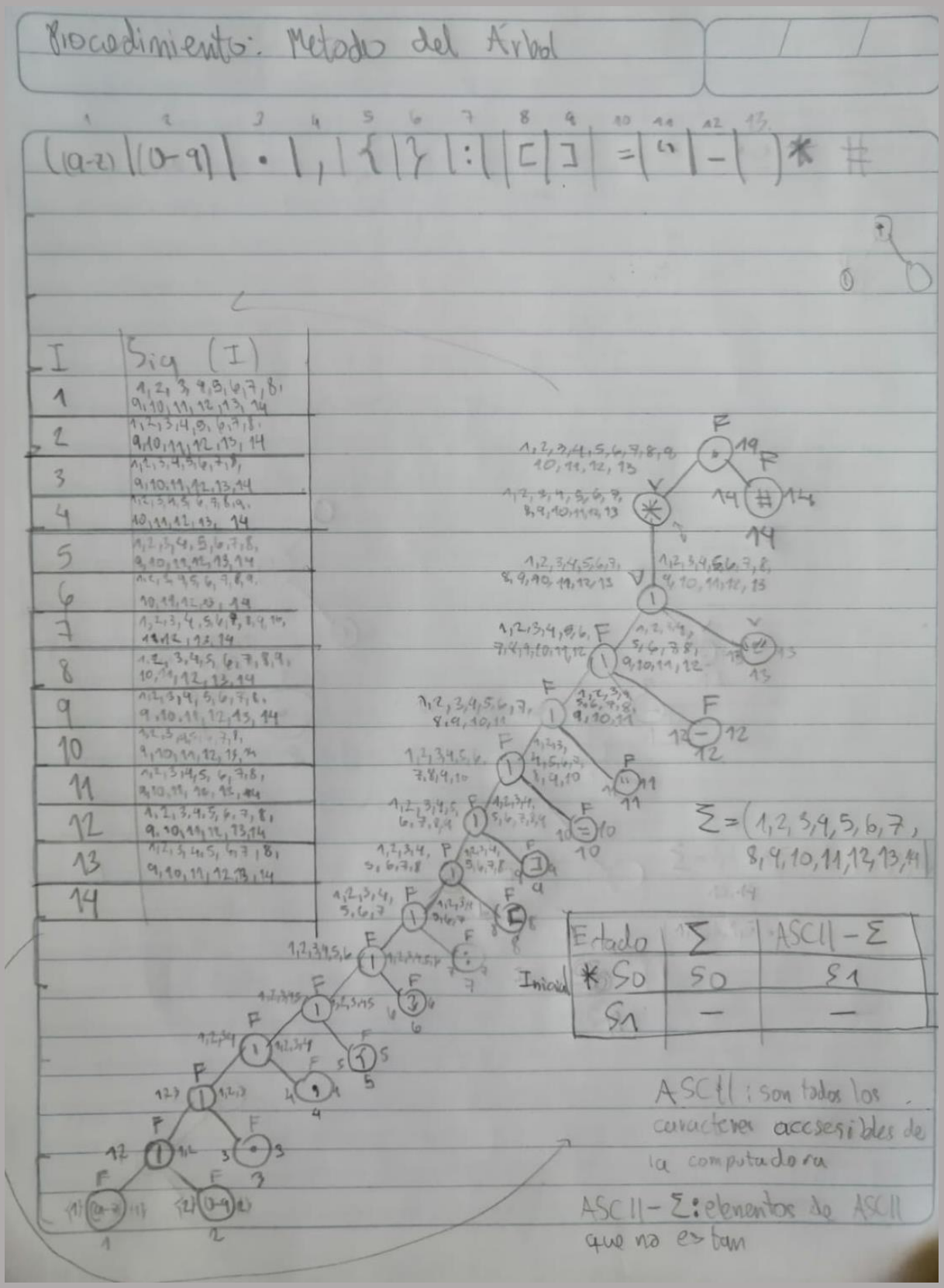
Se solicita la lectura de código fuente, el cual tendrá un formato JSON, creando un programa el cual sea capaz de identificar un lenguaje dado, identificando los errores léxicos y ejecutando las instrucciones correspondientes.
Se listarán una serie de instrucciones las cuales deben de ser ejecutadas, cumpliendo con el formato asignado, generándolo un resultado y graficarlos en un archivo según la jerarquía operacional de cada instrucción.

AFDs:

1. AFD de la revisión de errores léxicos (Scanner):

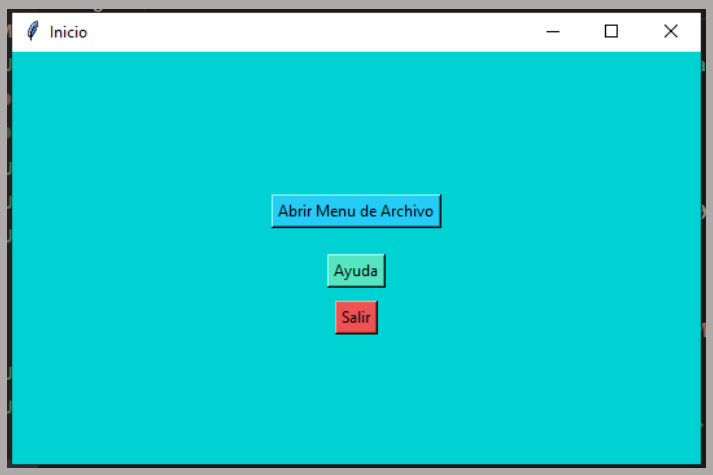


Procedimiento:

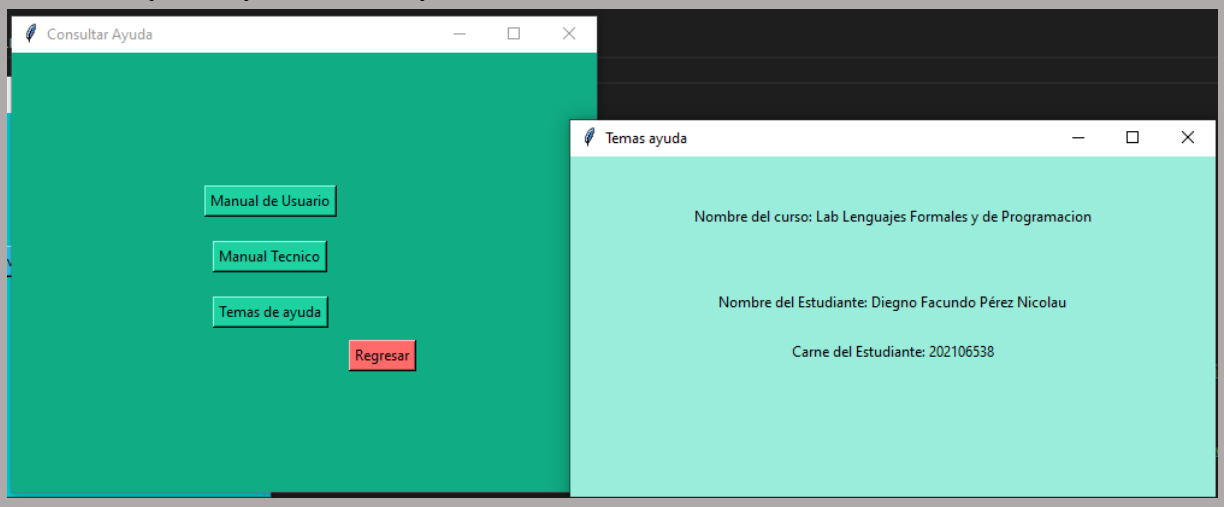


Interfaz:

Ventana inicio



Menú Ayuda y Temas Ayuda



Menú de Archivos