MANUAL TECNICO

PRACTICA 1

LENGUAJES DE PROGRAMACION

201904013
Marlon Isaí Figueroa Farfán

2021

INTRODUCCIÓN

Esta es una aplicación de escritorio desarrollada con el lenguaje de programación Python versión 3.8.1, utilizando como IDE Visual Studio Code.

Esta aplicación posee un menú de funcionalidades capaz de cargar archivos, leerlos, para posteriormente realizar un análisis y generar un resumen de datos en una página desarrollada con HTML implementando CSS y BOOSTRAP.

REQUERIMIENTOS DEL SISTEMA

SISTEMA OPERATIVO:

- MS Windows XP o superior.
- Apple OSX 10.4.x o superior.
- GNU/Linux 2.6.x o superior.

PROCESADOR:

- Procesador de 1,6 GHz o superior.

MEMORIA:

- 1 GB de RAM (1,5 GB si se ejecuta en una máquina virtual)

ESPACIO EN DISCO:

- 20 GB de espacio disponible en disco duro.
- Disco duro de 5400 RPM.

MÉTODOS Y PARADIGMAS UTILIZADOS

Las librerías utilizadas para este programa son:

```
#IMPORTANDO LIBRERIAS A UTILIZAR
import os
import tkinter.filedialog
from LISTA import lista
import webbrowser
```

MENÚ

Se creó un menú repetitivo para mostrar en pantalla con las siguientes opciones para el usuario:

```
def MENU():
    os.system('cls')
    print("************************************************************************")
    print ("\t          1 - Cargar archivo de entrada")
    print ("\t          2 - Desplegar listas ordenadas")
    print ("\t          3 - Desplegar búsquedas")
    print ("\t          4 - Desplegar todas")
    print ("\t          5 - Desplegar todas a archivo ")
    print ("\t          6 - Salir")
    print("************************************************************************")
```

Que independientemente de la opción que éste elija podrá volver fácilmente al menú de inicio para posteriormente elegir una nueva opción o salir del programa.

```
while True:
    MENU()
    op = input("\n ----Seleccione una Opción---- \n")
    if op=="1":
        print ("")
        Op1()
        input("\npulsa enter para volver...")
    elif op=="2":
        print ("")
        os.system('cls')
        Op2()
        input("\npulsa enter para volver...")
```

ABRIR VENTANA EMERGENTE

Para que el usuario pueda cargar su archivo con los datos se generó una ventana emergente utilizando la librería **tkinter** para que el mismo pueda buscar y abrir su archivo dentro del programa para posteriormente analizar los datos cargados.

```
def Op1():
    os.system('cls')
    print("Seleccione su archivo")
    global contenedor,NUMEROS,INSTRUCCIONES,DATOS,NOMBRES,DATOSIN
    global MOD1,MOD2,GENERAL
    root= tkinter.Tk()
    root.withdraw()
    ruta=tkinter.filedialog.askopenfilename(
        initialdir="C:",
        filetypes=(
            ("Ficheros de texto", "*.txt"),
            ("Todos los ficheros","*.*")
        ),
        title = "ABRIR ARCHIVO"
    )
```

LECTURA DE ARCHIVO TXT

Al momento de leer el archivo que previamente el usuario cargó se utilizó un **readlines()** para leer línea por línea y guardarla en una lista. Posteriormente las líneas fueron separadas por módulos para poder guardar los datos necesarios y realizar el análisis correspondiente.

```
try:
    #lectura de archivo linea*linea
    with open(ruta) as f:
        linea=f.readlines()

    #separando por espacios
    for i in linea:
        MOD1.append(i.split())
    #imprimiendo primer modulo
    #print("este es el modulo 1")
    #print(MOD1)
```

ORDENAMIENTO

Se utilizó el método de ordenamiento de burbuja donde podemos resaltar que (m, n, p) de la función **range** son el valor inicial, el límite (que no se alcanza nunca) y el paso (la cantidad que se avanza cada vez).

La condición de este ordenamiento es de que si la posición actual es mayor a la siguiente existe un cambio en la lista y así sucesivamente hasta tener toda la lista ordenada.

```
def ordenar(entrada):
    #ORDENAMIENTO DE BURBUJA
    for posicion in range(len(entrada)-1,0,-1):
        for i in range(posicion):
            #SI POSICION ACTUAL ES > A POSICION SIGUIENTE
            if entrada[i]>entrada[i+1]:
                temporal = entrada[i]
                entrada[i] = entrada[i+1]
                #SE REALIZA CAMBIO
                entrada[i+1] = temporal
```

CREAR REPORTE HTML

Para generar un reporte HTML se utilizó la función **f=open()** donde se indica la extensión y la acción que se realizará en el archivo a crear.

```
f = open('REPORTE.html','w')

f.write("""<!DOCTYPE html>
<html lang="en">
```

Luego de escribir los datos necesarios en el reporte se debe cerrar el documento con. **close()** para que posteriormente el documento se abra automáticamente usando la librería webbrowser.

```
</html>""")
f.close()

webbrowser.open_new_tab('REPORTE.html')
```

BÚSQUEDAS

Para realizar las búsquedas requeridas por el archivo que fue cargado previamente, se creó un método que recibe como parámetro una lista y el valor a buscar.

```
def comparar(lista,valor):
    global POSICIONES
    listemp=[]
    condicion=False
    if valor!="":
        temp=int(valor)
        for i in range(len(lista)):
            if lista[i]==temp:
                listemp.append(i+1)
                condicion=True
            else:
                condicion=False
        if condicion==False:
            return(print("--->no encontrado"))
        else:
            return(print(" --> encontrado en : " + str(listemp)))
    else:
        return(print(""))
```

Convierte el valor a entero para poder compararlo con los datos que contiene la lista. Si este es encontrado la función retorna las posiciones en que este se halló, si no la función retorna el mensaje de no encontrado dentro de la lista.

PROGRAMACÍON ORIENTADA A OBJETOS(POO)

El Paradigma de programación aplicado este programa fue la programación orientada objetos que ayudó a crear un modelo base para las listas.

```
class lista:

    def __init__(self,nombre,contenido,instrucciones,num):
        self.nombre = nombre
        self.contenido = contenido
        self.instrucciones = instrucciones
        self.num = num
```

Al momento de tener un modelo base o universal para las listas luego de haberlas separado por modulos se guardarón los datos correspondientes en un objeto, este nos permite tener un mejor manejo de datos dentro del programa.

```
for i in range(len(MOD1)):
    if len(MOD1[i])==2:
        INSTRUCCIONES.append(MOD1[i][1])
        NUMEROS.append("")
    elif len(MOD1[i])==3:
        INSTRUCCIONES.append(MOD1[i][1])
        NUMEROS.append(MOD1[i][2])

    else:
        print("instrucciones invalidas")
#print(INSTRUCCIONES)
#print("los numeros son:")
#print(NUMEROS)
GENERAL.append(lista(NOMBRES,DATOS,INSTRUCCIONES,NUMEROS))
```

CONCLUSIONES

- Se utilizaron métodos con parámetros para ahorrar procesos y memoria en el sistema
- La Programación orientada a objetos ayudó a tener un mejor control de datos y ahorrar código dentro del desarrollo del programa.
- Se utilizaron sentencias de control, ciclos y librerías externas para facilitar el desarrollo del programa.