Manual técnico

Practica Unica

Kelly Mischel Herrera Espino
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

Introducción

En el siguiente manual se describe el paradigma, las clases y los métodos utilizados en el programa. Con una explicación de la lógica aplicada en esta práctica. Así como lo que contiene cada clase y porque fue necesaria la creación de esta, para que se usó cada método y que pasaba al ejecutar el programa.

Manual técnico

Paradigma de programación orientada a objetos:

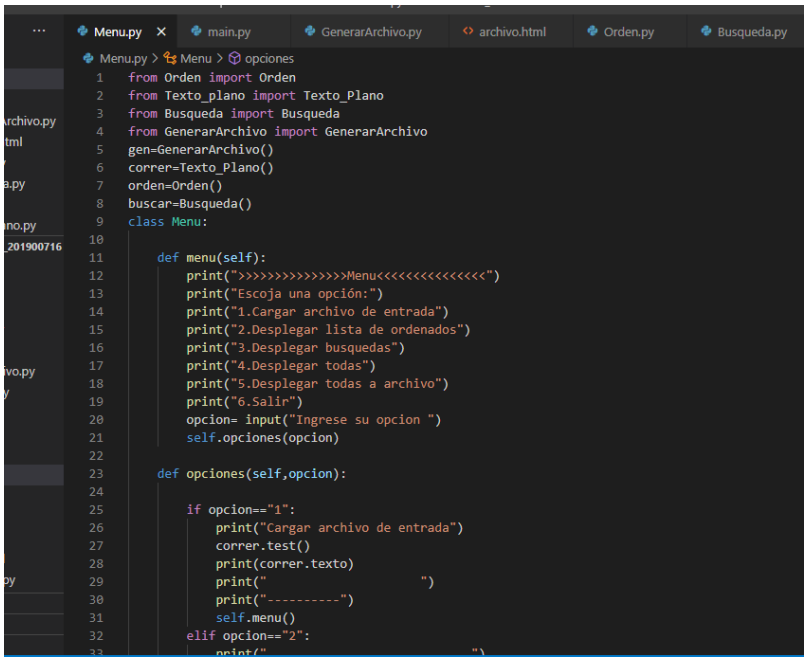
Se utilizo este paradigma para una mejor organización y abstracción de los datos. Para este programa fueron necesaria siete clases y una clase llamada Datos en donde se le asignaron 5 atributos que los objetos iban a tener en común. Con un arreglo de objetos se fueron guardando estos y poder obtener la referencia de estos en distintas clases. Con una clase con el método main ejecutar el programa.

Clase Menú

La clase menú contiene dos métodos con los datos que se le mostraran al usuario cuando corra el programa.

Los métodos son:

1. Menú
2. Opciones



Menú:

Este método con tiene varios mensajes que se imprimen en consola a llamar al método. También contiene la primera variable que almacenara el dato que ingrese el usuario en consola, la cual tiene por nombre opción. Por ultimo se llama el método opciones y se le coloca la variable opción en el parámetro.

```
def menu(self):
    print(">>>>>>>>>>>>>>>>Menu<<<<<<<<<<<<<<<<")
    print("Escoja una opción:")
    print("1.Cargar archivo de entrada")
    print("2.Desplegar lista de ordenados")
    print("3.Desplegar busquedas")
    print("4.Desplegar todas")
    print("5.Desplegar todas a archivo")
    print("6.Salir")
    opcion= input("Ingrese su opcion ")
    self.opciones(opcion)
```

Opciones:

En este método se colocan varias condiciones usando el if , elif y else para comparar el numero almacenado en la variable opción que el método recibió como parámetro. Si el numero es 1 entonces se llamar al método test de la clase Texto_Plano, si es 2 se llamar al método ordenamiento_bublesort de la clase Orden, si es 3 se llamara al método buscar_posicion de la clase Búsqueda, si la opción es 4 se llamara al método ordenamiento_bubblesor y al método buscar_posicion de sus clases respectivas. Si es 5 se llamará al método genara_archivo de la clase GeneraArchivo, si es 6 finalizara el programa y si en caso no es aluno de los números mencionados se volverá a mostrar el menú de opciones.

```
def opciones(self,opcion):

    if opcion=="1":
        print("Cargar archivo de entrada")
        correr.test()
        print(correr.texto)
        print("                    ")
        print("----------")
        self.menu()
    elif opcion=="2":
        print("                           ")
        print("----------------")
        print(">>>>>>>>>>>>>>Datos ordenados<<<<<<<<<<<<<<<<")
        orden.ordenamiento_bublesort()
        print("                                ")
        print("------------------")
        self.menu()

    elif opcion=="3":
        print(">>>>>>>>>Busqueda de posicones de dato<<<<<<<<<")
        buscar.buscar_posicion()
        print("busqueda finalizada")
        print("                        ")
        print("----------------------")
        self.menu()

    elif opcion=="4":
        print("                                    ")
        print(">>>>>>>>>>>>>>>>DESPLEGAR TODAS<<<<<<<<<<<<<<<<")
        orden.ordenamiento_bublesort()
        print("      ")
        buscar.buscar_posicion()
```

```
elif opcion=="4":
    print("                                  ")
    print(">>>>>>>>>>>>>>>DESPLEGAR TODAS<<<<<<<<<<<<<<<")
    orden.ordenamiento_bublesort()
    print("      ")
    buscar.buscar_posicion()
    print("        ")
    print("------------------")
    self.menu()
elif opcion=="5":
    print("                          ")
    print("Esocgio la opcion 5")
    gen.generar_archivo()
    print("Archivo generado ")
    self.menu()
elif opcion=="6":
    print("Programa Finalizado")
else:
    print("---------------------")
    print("Opcion erronea")
    self.menu()
```

Clase Búsqueda

Esta es una clase con dos métodos los cuales son:

Buscar_posicion:

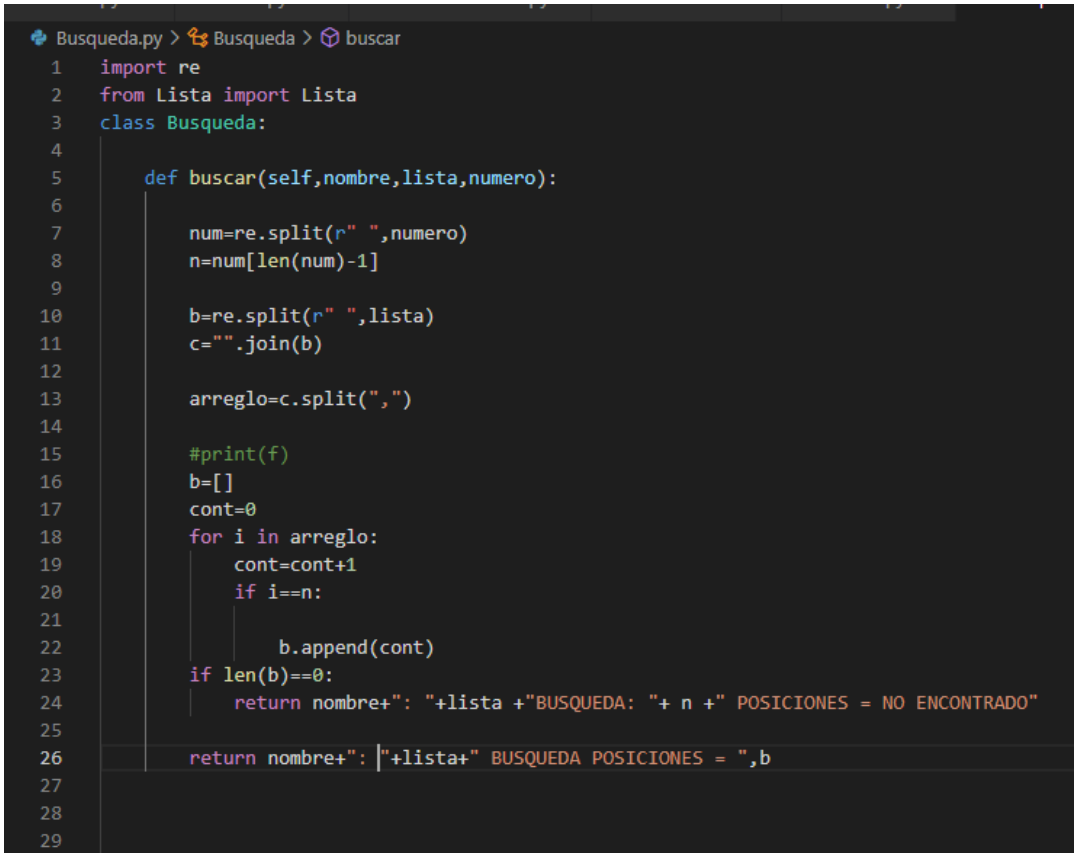
Este método con tiene un for que recorre el arreglo de objetos datos para poder obtener su valor. Dentro de este for se encuentra un if el cual compara si el atributo buscar tiene almacenado “BUSACAR” entonces que llame al método buscar y le mande dentro de los parámetros nombre, lista y numero de posición. Posterior a esto lo imprima en consola.

```
def buscar_posicion(self):

    for dato in Lista.datos:

        if dato.buscar=="BUSCAR":

            print(self.buscar(dato.nombre,dato.lista,dato.numero))

    return "."
```

Buscar:

Este método recibe como parámetro nombre que es el nombre de la lista, lista es la cadena de números y numero que es el dato que se desea buscar.

En este método primero se aplica un re.split a numero par poder quitarle los espacios que este contenga. Luego se almacena en la variable n. Luego a la cadena se procede a convertirla en una lista para poder aplicarle la búsqueda de las posiciones del dato n. Con un for se recorre la lista y un if dentro se coloca si el numero coincide con los que se obtiene en la lista entonces que se guarde cada posición en la que este se encuentra. Por ultimo si el numero no existe se retorna que no fue encontrado y si se encontró se retornan las posiciones.

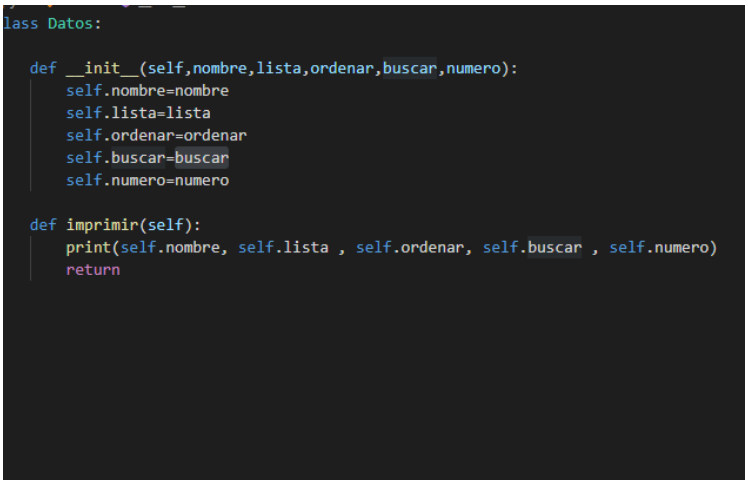


Clase Datos

En esta clase se le colocan los atributos que tiene el objeto junto con el respectivo constructor y un método para imprimir los datos.

Atributos:

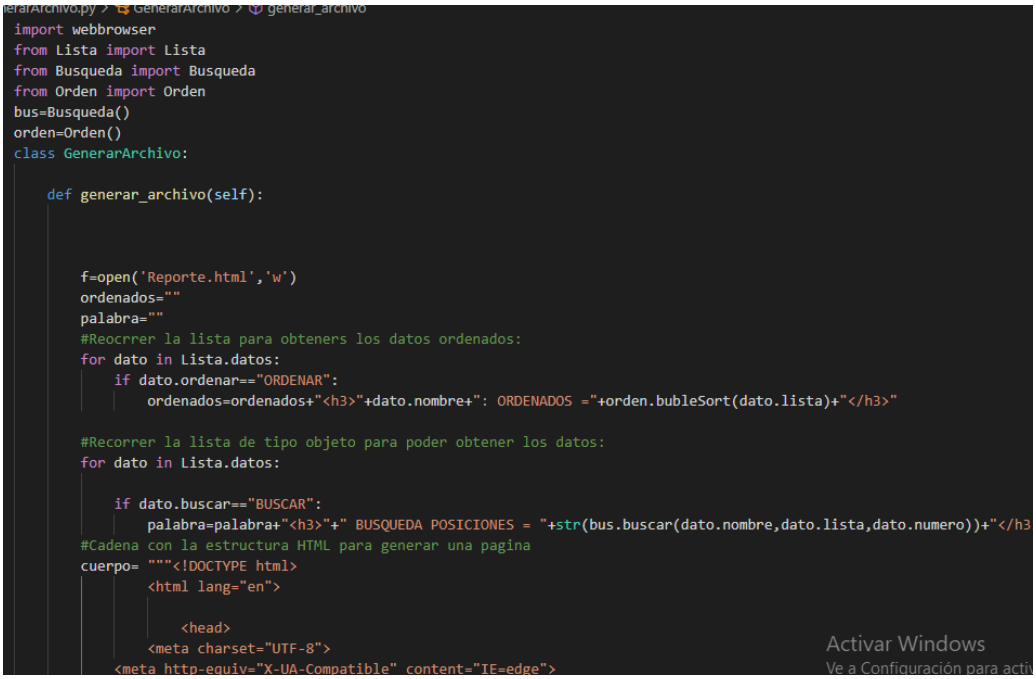
- **Nombre:** es el nombre de la lista
- **Lista:** son los datos de la lista
- **Ordenar:** la palabra Ordenar que acompaña a algunas listas.
- **Buscar:** la palabra “Buscar” que acompaña a ciertas listas.
- **Numero:** número a buscar.



```
lass Datos:

    def __init__(self,nombre,lista,ordenar,buscar,numero):
        self.nombre=nombre
        self.lista=lista
        self.ordenar=ordenar
        self.buscar=buscar
        self.numero=numero

    def imprimir(self):
        print(self.nombre, self.lista , self.ordenar, self.buscar , self.numero)
        return
```

Clase Generar_Archivo

En esta clase con un método se realiza el proceso de generar un archivo html. Importando webbrowser se procede a hacer uso de sus métodos como los el método open el cual recibe el nombre del archivo y la letra w para poder escribir lo que tendrá dentro. Luego con dos variables en la cuales se van a almacenar los datos del ordenamiento y la búsqueda. Con un for se procede a recorrer el arreglo de objetos para obtener la referencia. Por ultimo se coloca fi.write(cuerpo) para ecrbir lo que esta en la variable cuerpo dentro del archivo, luego fi.close() y en la ultima línea webbrowser.open_new_tab('Reporte.html').

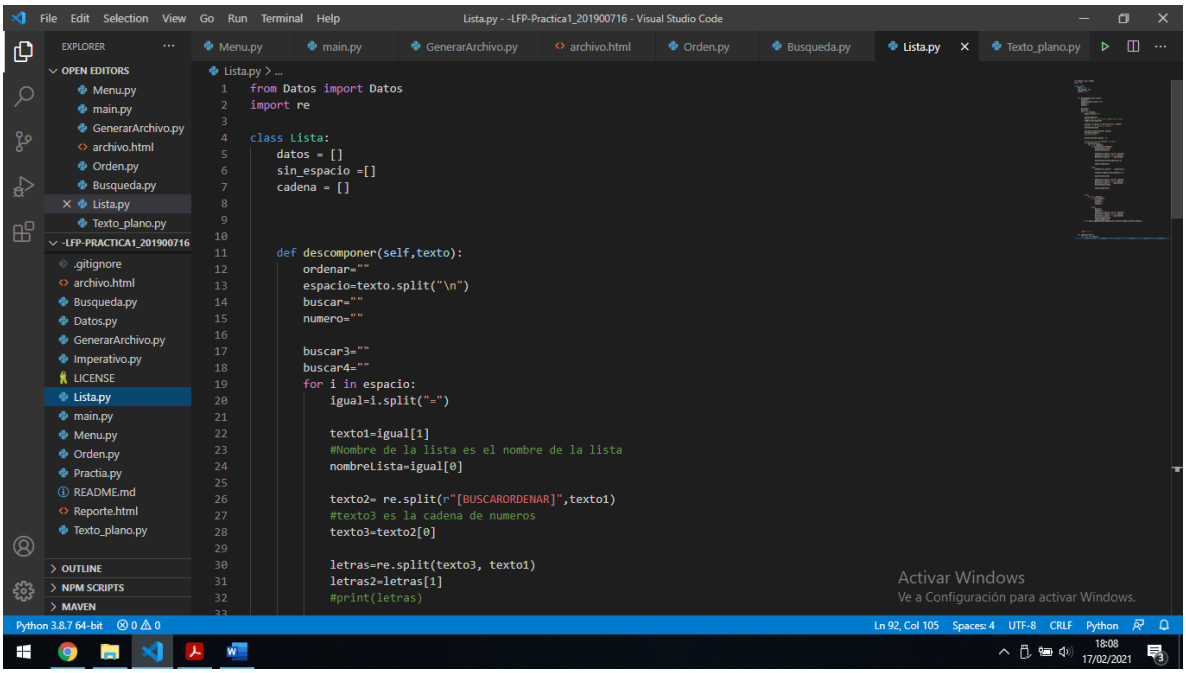


```python
import webbrowser
from Lista import Lista
from Busqueda import Busqueda
from Orden import Orden
bus=Busqueda()
orden=Orden()
class GenerarArchivo:

    def generar_archivo(self):



        f=open('Reporte.html','w')
        ordenados=""
        palabra=""
        #Reocrrer la lista para obteners los datos ordenados:
        for dato in Lista.datos:
            if dato.ordenar=="ORDENAR":
                ordenados=ordenados+"<h3>"+dato.nombre+": ORDENADOS ="+orden.bubleSort(dato.lista)+"</h3>"

        #Recorrer la lista de tipo objeto para poder obtener los datos:
        for dato in Lista.datos:

            if dato.buscar=="BUSCAR":
                palabra=palabra+"<h3>"+" BUSQUEDA POSICIONES = "+str(bus.buscar(dato.nombre,dato.lista,dato.numero))+"</h3
        #Cadena con la estructura HTML para generar una pagina
        cuerpo= """<!DOCTYPE html>
                <html lang="en">

                    <head>
                <meta charset="UTF-8">
            <meta http-equiv="X-UA-Compatible" content="IE=edge">
```


Clase Lista

Esta clase con tiene solo un método llamado descomponer en donde se procede a recibir el contenido del archivo. Por medio de un slplit se procede a quitar los salto de línea. Luego con un for para recorrer la cadena y un Split dentro se quita el signo igual para poder guardar en una variable la posición 0 de este arreglo.

Luego se realiza un re.split omitiendo lo que tenga la palabra BUSCAR y ORDENAR par luego almacenar los datos en una variable ya si pode obtener la cadena de números que formaran nuestra lista. Con un if se procede a compara el tamaño de la cadena que tiene almacenado los datos BUSCAR y ORDENAR para poder separarlos. Si el arreglo es de tamaño 2 entonces la lista tendrá las dos opciones ORDENAR y BUSCAR, como estas vienen en diferente orden se coloca un for dentro y un if para comparar si la primera es ORDENAR entonces se guardara con este nombre pero sino se guardar con la palabra “BUSCAR”. Ahora si el arreglo es de tamaño 1 entonces se procede a verificar si es BUSCAR u ORDENAR. Si fuera BUSCAR se coloca vacío en la la parte de buscar pero si es ORDENAR se coloca vacío en la variable buscar.




```
            else:
                ordenar1=re.split(r" ",buscar1[1])

                ordenar=ordenar1[len(ordenar1)-1]

                buscar=buscar1[0]

                numero1=re.split(r"BUSCAR",buscar)
                buscar2=re.split(r"[0-9]",buscar)
                buscar3=re.split(r" ",buscar2[0])
                buscar4=buscar3[0]

                numero=numero1[1]


    else:
        for l in buscar1:
            if l=="ORDENAR":
                ordenar=l
                buscar4=" "
                numero=" "

            else:
                buscar=l
                ordenar=" "
                numero1=re.split(r"BUSCAR",buscar)
                buscar2=re.split(r"[0-9]",buscar)
                buscar3=re.split(r" ",buscar2[0])
                buscar4=buscar3[0]
                numero=numero1[1]
    self.datos.append(Datos(nombreLista,texto3,ordenar,buscar4,numero))
```

Clase Orden

Esta clase contiene dos métodos los cuales son:

Bubblesor

Este método que recibe como parámetros una cadena, a esta cadena se procede a queitar los espacios que están se encuentren para luego convertirla en una lista de números enteros por medio de un for. Con el for se recorre la cadena y se almacena los datos en según su poscione en una lista conviritendolos en enteros lista.append(int(i)).

Para ordenar los datos se utilizó el método de burbuja colocando dos for que recorren la lista hasta llegar al tamaño de la lista por medio del método len y luego de range se pudo recorrer la lista siendo i la variable que se intervalo hasta llegar a el tamaño de la lista. Con un según for se uso la variable j y se recorrió la lista restándoles i y -1 al rango. Con un if dentro de estos dos fors se coloco que si el arreglo en la posición j+1 era menor al arreglo en la posición j entonces la variable temp seria igual al arreglo en su posición j, el arreglo en su posición j seria igual al arreglo en su posición j+ y por ultimo el arreglo en su posición j+1 seria igual a temp siendo esta la variable auxiliar.

Después de aplicar el ordenamiento bublesort se retornar el arreglo ordenado.

Ordenamiento_bubblesort:

Dentro de este método se recorre el arreglo de objetos para comparar si las listas se deben de ordenar o no. Dentro de un for se coloca un if que usa la condición mencionada anteriormente y si cumple con esta llama al método bubblesort y le manda como parámetro dato.lista. Al finalizar este proceso imprime en consola los numero ordenados.

```
        return arreglo2

    #Metodo para obtener los datos del arreglo de objetos Datos.
    def ordenamiento_bublesort(self):
        for dato in Lista.datos:
            if dato.ordenar=="ORDENAR":
                print(dato.nombre+": ORDENADOS ="+self.bubleSort(dato.lista))

        return "."
```

Clase Texto_Plano

En esta clase se abre un ventana en donde se puede escoger el archivo que se desea cargar des cualquier directorio de la computadora. Usando el método .redad para leer lo que esta adentro del archivo.

Texto_plano.py > Texto_Plano > test

```
from tkinter import filedialog as FileDialog
from Lista import Lista

lista=Lista()

class Texto_Plano:

    texto=0

    def test(self):
        fichero= FileDialog.askopenfilename(title="Abrir un fichero")
        fi=open(fichero,'r')
        mensaje=fi.read()
        self.texto=mensaje
        lista.descomponer(mensaje)




lista.imprimir()
```

Conclusión

El programa se logro estructurar y acoplar a los requerimientos que se pedían por parte del cliente. Utilizando el paradigma de programación orientada a objetos se obtuvo una mejor organización del código que hacía más eficaz con una mejor organización a la hora de llamar clases y métodos. Debido a que se tiene un mejor orden y percepción de lo que se busca realizar o no. Para llegar a obtener un programa eficiente.