Universidad De San Carlos de Guatemala   
Facultad de Ingeniería  
Escuela de Ciencias y Sistemas

Lenguajes Formales y de Programación  
Sección “B-”

**“MANUAL TÉCNICO”**

**Carlos Augusto Calderón Estrada**

**201905515**

**Objetivos**

**General**:

Brindarle al desarrollador a trabajar en esta aplicación una guía lo más completa y sencilla posible para el mejor entendimiento de lo que se lleva desarrollado en la aplicación y así evitar errores posibles por el uso incorrecto de los algoritmos y/o estructuras.

**Específicos**:

* Que el desarrollador trabaje en la aplicación como si fuera creada por el/ella.
* Proporcionarle al lector una explicación sencilla y técnica de los procesos algorítmicos y las relaciones de los métodos, funciones y atributos que son esenciales en la aplicación.

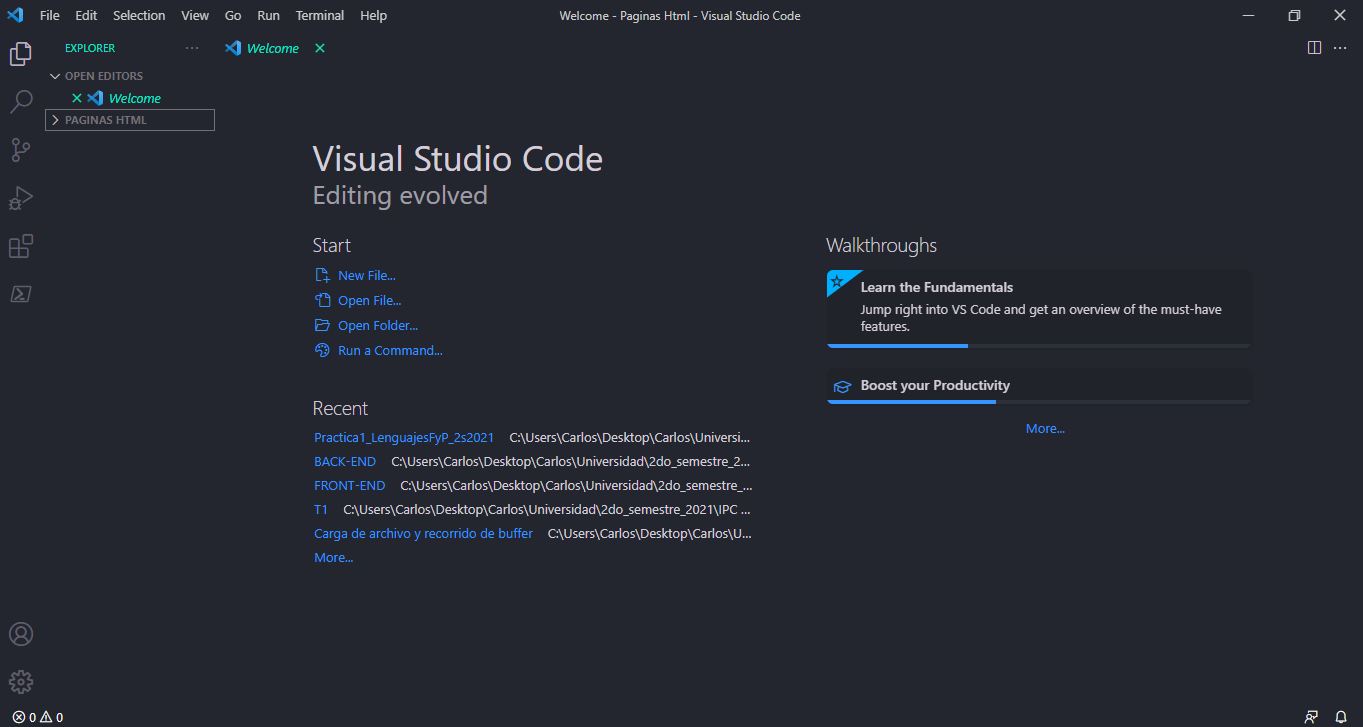
**Introducción**

Este manual técnico tiene como fin dar a conocer al desarrollador las mejores recomendaciones, requerimientos para un mejor entendimiento al momento de querer realizar modificaciones, optimización de procesos o métodos, indicando el editor de texto que se utilizó para el desarrollo de dicha aplicación.

La aplicación se basa en Analizar un archivo por medio de un autómata finito creado por el desarrollador, en el que se cargara por medio de una ventana emergente y este carga el archivo al programa(todo esto a través de una interfaz gráfica de Python) y se procede a realizar el análisis de dichos datos a utilizar en dicha ejecución, en el que por medio de estructuras de control **IF**, que cualquier desarrollador con un poco de experiencia conoce, se basa el autómata, almacenándolas en variables que serán utilizadas como tokens y son indispensables, luego de realizar el análisis, se generaran html que contendrán un lienzo o cuadro de pixels, estos contendrán un estilo único para cada imagen, y por medio de graphviz generamos la imagen que se visualiza en html, estas mismas podrán ser visualizadas en el apartado de Analizar, con finalidad se generan los reportes de errores y de tokens por medio de escritura de archivo que contienen la lista de errores y de tokens.

**Editor de texto utilizado**

1. El editor de texto utilizado fue Visual Studio Code gracias a la variedad de extensiones que posee y su fácil uso para el manejo de la nube git, que permite agregar la propia terminal y ejecutar los comandos desde el entorno de VS Code.



**Requerimientos:**

* Procesador de 1.6 GHz o más.
* 1 GB de memoria RAM
* Plataformas aprobadas: OS X, Windows 7, Linux(Debian), Linux (Red Hat)
* Microsoft .NET Framework 4.5.2 para VS Code

**Sistema Operativo que se llevó a cabo:**

Windows 10 de 64 bits

**Diagrama de flujo**

**Diagrama

Descripción generada automáticamente**

**Lógica del Programa**

**Menú:**

print("========================= Bienvenido a la practica 1 de Lenguajes FYP =========================")

NombreCurso=""

parametro=""

gestor = Gestor()

def Menu():

    print("\n =============Menu=============")

    print("1. Cargar Archivo")

    print("2. Mostrar reporte en consola")

    print("3. Exportar reporte")

    print("4. Salir")

    opcion = int(input("Ingrese la opcion a realizar: "))

    while opcion != 4:

        if opcion == 1:

            txto = CargarArchivo()

            if txto is not None:

                print("Texto almacenado")

            else:

                print("error de lectura")

            break;

        elif opcion == 2:

            MostrarReporte()

            Menu()

            break;

        elif opcion == 3:

            ExportarReporte()

            Menu()

            break

        else:

            print("Escoja una de las opciones que se le presentan ")

            Menu()

**Cargar Archivo:**

def CargarArchivo():

    Tk().withdraw()

    archivo = filedialog.askopenfile(initialdir="./Archivos prueba", title="Seleccione un archivo",filetypes=(("LFP files",".lfp"),("ALL files",".txt")))

    if archivo is None:

        print('No se selecciono ni un archivo\n')

        return None

    else:

        global NombreCurso

        global parametro

        parametro =""

        texto = archivo.read()

        NombreCurso = ObNombreCurso(texto)

        terminanEstudiantes = AddEstudiantes(texto)

        parametro = ObtenerParametro(texto, terminanEstudiantes)

        print("Los parametros son: "+parametro)

        archivo.close()

        print('Lectura exitosa\n')

        Menu()

**Obtener nombre Curso:**

def ObNombreCurso(texto):

    NombreCurso = ""

    i=0

    for x in texto:

        if x == "=":

            i = i+1

            return NombreCurso

        else:

            NombreCurso = NombreCurso + texto[i]

            i=i+1

**Agregar Estudiantes a Lista:**

def AddEstudiantes(texto):

    i=0

    while texto[i] != '{':

        i+=1

    while texto[i]!='}':

        Nota = ''

        i+=1

        while texto[i] !='<':

            i+=1

        i+=1

        while texto[i] !='"':

            i+=1

        i+=1

        NombreEst =""

        while texto[i] != '"':

            NombreEst = NombreEst + texto[i]

            i+=1

        i+=1

        while texto[i] != ';':

            i+=1

        i+=1

        while texto[i] != '>':

            if texto[i] != " ":

                Nota = Nota + texto[i]

                i+=1

            else:

                i+=1

        notaEnviar = abs(int(Nota))

        i+=1

        while texto[i] != ',':

            if texto[i] =='}':

                gestor.AgregarEstudiante(NombreEst,notaEnviar)

                break

            else:

                i+=1

        if texto[i]==',':

            gestor.AgregarEstudiante(NombreEst,notaEnviar)

    return i

**Obtener Parámetro:**

def ObtenerParametro(texto, i):

    parametro= ''

    while texto[i] != '}':

        i+=1

    i+=1

    while texto[i]==" ":

        i+=1

    while texto[i] != ",":

        if texto[i]!="":

            parametro = parametro+texto[i]

            i+=1

            if i ==len(texto):

                return parametro

1. **Mostrar Reporte:**

def MostrarReporte():

    global NombreCurso

    global parametro

    print("============="+NombreCurso+"=============")

    gestor.ImprimirEstudiantes(parametro)

**Imprimir Estudiantes:**

for x in parametros:

            if x == ",":

                contadorParametros=contadorParametros+1

        print("Estudiantes ingresados")

       #Uso de libreria para una mejor estructura de la lista en consola

        a = PrettyTable()

        a.field\_names = ["Estudiante", "Nota"]

        for x in self.Estudiante:

            if x.NombreAlumno != " ":

                a.add\_row([x.NombreAlumno, x.NotaAlumno])

        print(a)

**Gestor para imprimir los parámetros solicitados:**

i=0

        aux1Parametros = parametros.replace(" ","")

        aux2Parametros = aux1Parametros.replace("\n","")

        for x in aux2Parametros:

            if i < len(aux2Parametros):

                parametro =""

                while aux2Parametros[i] != "," :

                    parametro = parametro+aux2Parametros[i]

                    if i+1 < len(aux2Parametros):

                        i+=1

                    else:

                        if i == len(aux2Parametros):

                            break

                        else:

                            i+=1

                            break

                i+=1

                if parametro == "AVG":

                    pidio\_Parametro =True

                    pidioPromedio =True

                    self.ObtenerPromedio()

                elif parametro =="MAX":

                    pidio\_Parametro =True

                    pidio\_NotaMax =True

                    self.ObtenerNotaMax()

                elif parametro == "MIN":

                    pidio\_Parametro =True

                    pidio\_NotaMin =True

                    self.ObtenerNotaMinima()

                elif parametro =="ASC":

                    pidio\_Parametro =True

                    pidio\_ASC=True

                    self.OrdenarAsc()

                elif parametro == "DESC":

                    pidio\_Parametro =True

                    pidio\_DESC =True

                    self.OrdenarDESC()

                elif parametro == "APR":

                    pidio\_Parametro =True

                    pidio\_APR = True

                    self.ObtenerAprobados()

                elif parametro == "REP":

                    pidio\_Parametro =True

                    pidio\_REP = True

                    self.ObtenerReprobados()

                else:

                    print("El parametro no existe")

            else:

                break

**Imprimir los Datos estadísticos:**

 self.GestorParametros(parametros)

        if pidioPromedio == TRUE:

            print("El promedio es: "+ str(promedio))

        if pidio\_NotaMax == TRUE:

            print("La nota maxima es: "+ str(NotaMax))

        if pidio\_NotaMin == TRUE:

            print("La nota minima es: "+ str(NotaMinima))

        if pidio\_APR == TRUE:

            print("La cantidad de aprobados es: "+ str(ContadorAprobados))

        if pidio\_REP == TRUE:

            print("La cantidad de reprobados es: "+ str(ContadorReprobados))

**Reporte HTML:**

def ExportarReporte():

    global NombreCurso

    gestor.ExportarReporte(NombreCurso)

**Ingresar datos de lista en una tabla HTML:**

i=0

        for x in self.Estudiante:

            if x.NombreAlumno!="" and x.NotaAlumno != "":

                i+=1

                if x.NotaAlumno < 61:

                    textoTabla=textoTabla+'<tr>'+'<th scope="row">'+str(i)+'</th>'+'<td>'+x.NombreAlumno+'</td>'+'<td style="color:red;">'+str(x.NotaAlumno)+'</td>'+'</tr>'

                else:

                    textoTabla=textoTabla+'<tr>'+'<th scope="row">'+str(i)+'</th>'+'<td>'+x.NombreAlumno+'</td>'+'<td style="color:blue;">'+str(x.NotaAlumno)+'</td>'+'</tr>'

**Crear El archivo HTML:**

file = open("./REPORTE/Reporte.html","w")

        file.write(str(contenidoHTML))

**Librerías Utilizadas**

Las librerías que se utilizaron para el desarrollo de esta practica fueron:

**Imports locales:**

* from Token import Token
* from Errores import Errores
* from Imagen import Imagen
* from Celda import Celda
* from Gestor import \*

**Imports TKINTER:**

Estas librerías fuerin utilizadas para la carga del archivo

* from tkinter import \*
* from tkinter import filedialog, Tk
* from tkinter.filedialog import askopenfilename
* from graphviz import \*

**Imports Interfaz PyQt5:**

Estas librerías fueron utilizadas para la creación de la interfaz gráfica.

* from PyQt5 import QtCore, QtGui, QtWidgets

**Descripción de Tokens**

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Token** | **Descripción** | **Tipo de valor** |
| **1** | Titulo | Este token representa el título de la imagen | String |
| **2** | Ancho | Este token representa el ancho del Lienzo o cuadro de pixeles | Int  [0,9] |
| **3** | Alto | Este token representa el alto del lienzo o cuadro de pixeles | Int  [0,9] |
| **4** | Filas | Este token representa el número de filas del lienzo o cuadro de pixeles | Int  [0,9] |
| **5** | Columnas | Este token representa el número de columnas del lienzo o cuadro de pixeles | Int  [0,9] |
| **6** | X | Este token representa la posición del píxel en columna del lienzo o cuadro de pixeles | Int  [0,9] |
| **7** | Y | Este token representa la posición del píxel en fila del lienzo o cuadro de pixeles | Int  [0,9] |
| **8** | ValorBool | Este token representa si la celda ubicada será pintada | Booleano  (true o false) |
| **9** | Color | Este token representa el color con el que será pintado el píxel | Color tipo Hexadecimal  ejemplo(#F0F8FF) |