UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

LENGUAJES FORMALES Y DE PROGRAMACIÓN

SEGUNDO SEMESTRE

MANUAL

-

TÉCNICO

TABLA DE CONTENIDO

LICENCIA DEL DESARROLLADOR

CHRISTOPHER IVÁN MONTERROSO ALEGRIA.

CARNÉ:201902363

ESTUDIANTE INGENIERIA EN CIENCIAS Y SISTEMAS.

REQUSISITOS DE LA APLICACIÓN:

- **WINDOWS 10,8,7(X64, X32).**
- **PROCESADOR 1.4GHZ O SUPERIOR**
- **2 GB RAM MINIMO, 4 RECOMENDADO**
- **500 MB ESPACIO LIBRE**
- **PYTHON 2.7 O SUPERIOR – HTTPS://WWW.PYTHON.ORG/DOWNLOADS/**
  VER MANUAL DE INSTALACIÓN EN LA PÁGINA OFICIAL

EJECUCIÓN DEL PROGRAMA.

MÉTODO VENTANA PRINCIPAL:

ESTE MÉTODO GENERA LA VENTANA PRINCIPAL CON SUS COMPONENTES Y FUNCIONES.

```
    def __init__(self,root) -> None:
        self.ventana=root
        self.ventana.title("Analizador léxico")
        self.ventana.configure(bg="white")
        self.ventana.iconbitmap(".\Icono\icono.ico")
        self.ventana.resizable(0,0)
        self.gadgets()

    def gadgets(self):···
    def Salir (self):···
    def Abrir_Archivo(self):···
    def analizar(self):···

    def Guardar(self):···

    def Guardar_como(self,file_path=None):···

    def B_Ayuda(self):···

    def Abrir_Html_Errores(self):···
    def Abrir_Html_Resultados(self):···
```

MÉTODO ABRIR ARCHIVO:

ESTE MÉTODO PERMITE QUE EL PROGRAMA ABRA UNA VENTANA EMERGENTE PARA SELECCIONAR UN ARCHIVO CON LA LIBRERÍA EASYGUI

```
def Abrir_Archivo(self):
    self.ruta = easygui.fileopenbox(title="Abre el archivo .txt")
    try:
        archivo= open(self.ruta,'r',encoding='utf-8')
        contenido=archivo.read()
        archivo.close()
    except:
        print("ERROR primer try")
    try:
        self.MostrarTxt.delete("1.0",END)
        self.MostrarTxt.insert(END,contenido)
    except:
        print("ERROR")
```

MÉTODO GUARDAR COMO:

ESTE MÉTODO PERMITE QUE EL PROGRAMA GENERE UN ARCHIVO NUEVO A BASE DE LO ENCUENTRA EN EL CUADRO DE TEXTO.

```
def Guardar_como(self,file_path=None):
    contenido= str(self.MostrarTxt.get("1.0",END))
    if contenido!="":
        if file_path is None:
            try:
                file_path = filedialog.asksaveasfilename(
                    filetypes=(("Archivos de texto", "*.txt"),("Todos los archivos", "*.*")))
                ruta= open(file_path,'w')
                ruta.write(contenido)
                ruta.close()
            except:
                print("ERROR")
```

MÉTODOS DE LA DOCUMENTACIÓN Y HTML:

ESTOS MÉTODOS ABREN EN EL NAVEGADOR PREDETERMINADO EL ARCHIVO QUE SE LE INDICA AL PROGRAMA PRESIONANDO EL BOTÓN ABRIR PARA LOS PDF Y ERRORES O RESULTADOS PARA EL BOTÓN ERROR Y ANALIZAR RESPECTIVAMENTE.

```
def B_Ayuda(self):
    eleccion= self.Boton_Ayuda.get()
    if eleccion=="Manual de usuario":
        webbrowser.open_new_tab("file:///"+os.getcwd()+"/Manuales/Manual de Usuario.pdf")
    elif eleccion=="Autor":
        webbrowser.open_new_tab("file:///"+os.getcwd()+"/Manuales/Autor.pdf")
    elif eleccion=="Manual técnico":
        webbrowser.open_new_tab("file:///"+os.getcwd()+"/Manuales/Manual técnico.pdf")

def Abrir_Html_Errores(self):
    print(os.getcwd())
    if os.path.exists("./Records/ERRORES_201902363.html"):
       webbrowser.open_new_tab("file:///"+os.getcwd()+"/Records/ERRORES_201902363.html")
    else:
        messagebox.showinfo(title="Atención",message="No se ha encontrado el archivo")
def Abrir_Html_Resultados(self):
    if os.path.exists("./Records/RESULTADOS_201902363.html"):
        webbrowser.open_new_tab("file:///"+os.getcwd()+"/Records/RESULTADOS_201902363.html")
    else:
        messagebox.showinfo(title="Atención",message="No se ha encontrado el archivo")
```

MÉTODO ANALIZAR:

ESTÉ MÉTODO EJECUTA EL ANALIZADOR DE TEXTO.

```
def analizar(self):
    contenido= str(self.MostrarTxt.get("1.0",END))
    if contenido!="":

        Analizador(contenido).compile()
        self.Abrir_Html_Resultados()
```

MÉTODOS PARA LIMPIAR EL TEXTO ENTRANTE.

```
def quitar(self, _cadena :str, _num : int):

    _tmp = ""
    count = 0
    for i in _cadena:
        if count >= _num:
            _tmp += i
        else:
            self.tmp_cadena += i
        count += 1

    return _tmp

def aumentarLinea(self):
    _tmp = self.lista_cadena[self.linea]

    if _tmp == self.tmp_cadena:
        self.linea += 1
        self.tmp_cadena = ""
        self.columna = 0
```

```
def esLaetiqueta(self, _cadena : str, _etiqueta : str):
    tmp = ""
    count = 0
    for i in _cadena:
        if count < len(_etiqueta):
            tmp += i
        count += 1

    if tmp == _etiqueta:
        return True
    else:
        return False
```

ESTE MÉTODO LEE LÍNEA POR LÍNEA LA ETIQUETA NÚMERO DEL TEXTO ENTRANTE, Y GUARDANDO EN UNA LISTA LOS NÚMEROS PARA LA ETIQUETA TEXTO Y EN DADO CASO ENCUENTRA UN ERROR LO GUARDA EN UNA LISTA.

```python
def Numero(self, _cadena : str):
    tokens = [ ···
    _numero = ""

    for i in tokens:
        try:
            patron = re.compile(f'^{i}')
            s = patron.search(_cadena)
            print("| ", self.linea, " | ", self.columna, " | ", s.group())
            self.columna += int(s.end())
            # GUARDAR EL TOKEN
            if i == L_tokens.TK_NUMERO.value:

                _numero = s.group()
                self.lis.append(float(_numero))
            _cadena = self.quitar(_cadena, s.end())
            self.aumentarLinea()
        except:
            # GUARDAR ERROR
            print("GUARDANDO ERROR")
            e = err(i, self.linea, self.columna, "Error de código")
            listaErroresLexicos.append(e)
            self.reiniciar()
            print("Ocurrio un error")
            return {'resultado':_numero, "cadena":_cadena, "Error": True}
```

ESTE MÉTODO LEE LÍNEA POR LÍNEA LA ETIQUETA OPERACIÓN DEL TEXTO ENTRANTE, Y GUARDANDO EN UNA LISTA LAS OPERACIONES PARA LA ETIQUETA OPERACIÓN Y EN DADO CASO ENCUENTRA UN ERROR LO GUARDA EN UNA LISTA.

```python
def Operacion(self, _cadena : str):
    tokens = [ ···
    _numero = ""
    _operador = None
    for i in tokens:
        try:
            if "NUMERO" == i:
                if self.esLaetiqueta(_cadena, "<Numero>"):
                    _result = self.Numero(_cadena)
                    _cadena = _result['cadena']
                    if _result['Error']:
                        # GUARDAR ERROR
                        e = err(i, self.linea, self.columna, "Error de código")
                        listaErroresLexicos.append(e)
                        self.reiniciar()

                        print("Ocurrio un error")
                        return {'resultado':_numero, "cadena":_cadena, "Error": True}

                elif self.esLaetiqueta(_cadena, "<Operacion="):
                    _result = self.Operacion(_cadena)
                    _cadena = _result['cadena']
                    if _result['Error']:
                        # GUARDAR ERROR
                        e = err(i, self.linea, self.columna, "Error de código")
                        listaErroresLexicos.append(e)
                        self.reiniciar()
                        print("Ocurrio un error")
                        return {'resultado':_numero, "cadena":_cadena, "Error": True}
                else:
                    # GUARDAR ERROR
                    e = err(i, self.linea, self.columna, "Error de código")
                    listaErroresLexicos.append(e)
                    self.reiniciar()
                    print("Ocurrio un error")
                    return {'resultado':_numero, "cadena":_cadena, "Error": True}
```

ESTE MÉTODO LEE LÍNEA POR LÍNEA LA ETIQUETA TIPO DEL TEXTO ENTRANTE, SI EN DADO CASO ENCUENTRA UN ERROR LO GUARDA EN UNA LISTA.

```
def Tipo(self, _cadena : str):

    tokens = [ ···
    _numero = ""

    for i in tokens:
        try:
            if "OPERACIONES" == i:
                salida = True
                while salida:
                    print("--------------------------------")
                    self.lis = []
                    _result = self.Operacion(_cadena)
                    lista_total.append(self.lis)
                    _cadena = _result['cadena']
                    if _result['Error']:
                        # GUARDAR ERROR
                        e = err(i, self.linea, self.columna, "Error de código")
                        listaErroresLexicos.append(e)
                        self.reiniciar()
                        print("Ocurrio un error")
                        salida=False
                    if self.esLaetiqueta(_cadena, "</Tipo>"):
                        salida = False
            else:
                patron = re.compile(f'^{i}')
                s = patron.search(_cadena)
                # GUARDAR EL TOKEN

                print("| ", self.linea, " | ", self.columna, " | ", s.group())
                self.columna += int(s.end())
                _cadena = self.quitar(_cadena, s.end())

            self.aumentarLinea()
        except:
```

ESTE MÉTODO LEE LÍNEA POR LÍNEA LA ETIQUETA TEXTO Y ALMACENA SU CONTENIDO EN UNA VARIABLE TIPO STRING.

```
def Texto(self, _cadena : str):
    tokens = [ ···
    _numero = ""
    for i in tokens:
        try:
            patron = re.compile(f'^{i}')
            s = patron.search(_cadena)
            print("| ", self.linea, " | ", self.columna, " | ", s.group())
            self.columna += int(s.end())
            # GUARDAR EL TOKEN
            if i == L_tokens.TK_E_PARRAFO.value:
                self.txt += s.group()
            _cadena = self.quitar(_cadena, s.end())
            self.aumentarLinea()
        except:
            # GUARDAR ERROR
            e = err(i, self.linea, self.columna, "Error de codigo")
            listaErroresLexicos.append(e)
            self.reiniciar()
            print("Ocurrio un error")

            return {'resultado':_numero, "cadena":_cadena, "Error": True}
    self.Funcion(_cadena)
    return {'resultado':_numero, "cadena":_cadena, "Error":False}
```

ESTE MÉTODO LEE LÍNEA POR LÍNEA LA ETIQUETA FUNCIÓN Y ALMACENA SU CONTENIDO EN UNA VARIABLE TIPO STRING.

```
def Funcion(self, _cadena : str):
    tokens = [ ···
    txt=""
    for i in tokens:
        try:
            patron = re.compile(f'^{i}')
            s = patron.search(_cadena)
            print("| ", self.linea, " | ", self.columna, " | ", s.group())
            self.columna += int(s.end())
            # GUARDAR EL TOKEN
            if i == L_tokens.TK_E_PARRAFO.value:
                txt += s.group()

            _cadena = self.quitar(_cadena, s.end())
            self.aumentarLinea()
        except:
            # GUARDAR ERROR
            e = err(i, self.linea, self.columna, "Error de codigo")
            listaErroresLexicos.append(e)
            self.reiniciar()
            print("Ocurrio un error")
            return { "cadena":_cadena, "Error": True}
    self.titulo=txt
    self.Estilo(_cadena)
    return { "cadena":_cadena, "Error":False}
```

ESTE MÉTODO LEE LÍNEA POR LÍNEA LA ETIQUETA ESTILO Y ALMACENA LOS COLORES Y TAMAÑOS EN UNA LISTA.

```
    def Estilo(self,_cadena : str):
        tokens =[
            "<","Estilo",">",
            "<","Titulo","Color","=",L_tokens.TK_MAYUS.value,"tamanio","=",L_tokens.TK_NUMERO.value ,"/",">"
            ,"<","Descripcion","Color","=",L_tokens.TK_MAYUS.value,"tamanio","=",L_tokens.TK_NUMERO.value,"/",">",
            "<","Contenido","Color","=",L_tokens.TK_MAYUS.value,"tamanio","=",L_tokens.TK_NUMERO.value,"/",">"
            "<","/","Estilo",">"
        ]


        for i in tokens:
            colores=""
            tamaños=0
            listaaux=[]
            try:
                patron = re.compile(f'^{i}')
                s = patron.search(_cadena)
                print("| ", self.linea, " | ", self.columna, " | ", s.group())
                self.columna += int(s.end())
                # GUARDAR EL TOKEN
                if i == L_tokens.TK_MAYUS.value:
                    colores += s.group()
                if colores!="":
                    self.colores.append(colores)
                if i == L_tokens.TK_NUMERO.value:
                    tamaños = int(s.group())

                if tamaños>0:
                    self.tamaños.append(tamaños)
                _cadena = self.quitar(_cadena, s.end())
                self.aumentarLinea()
            except:
                # GUARDAR ERROR
                self.reiniciar()
                e = err(i, self.linea, self.columna, "Error de codigo")
                listaErroresLexicos.append(e)
```

ESTE MÉTODO EJECUTA EN ORDEN LOS MÉTODOS ANTERIORMENTE MOSTRADOS PARA LA LECTURA DEL TEXTO DE ENTRADA

```
def compile(self):
    # LIMPIAR MI ENTRADA
    nueva_cadena = ""
    lista_cadena = []
    for i in self.contenido:
        i = i.replace(' ', '') #QUITANDO ESPACIOS
        i = i.replace('\t', '')
        i = i.replace('\n', '') # QUITANDO SALTOS DE LINEA
        if i != '':
            nueva_cadena += i
            lista_cadena.append(i)
    self.lista_cadena = lista_cadena
    self.Tipo(nueva_cadena)
    print(listaErroresLexicos)
    print(lista_total)
    if listaErroresLexicos:
        self.report.Errores(listaErroresLexicos)
    if len(self.tamaños)>0:
        calculo(lista_total,self.txt,self.titulo,self.colores,self.tamaños)
        self.reiniciar()
    else:
        messagebox.showerror(title="Atención",message="Se detectó un error, presione el botón errores para mostrar el error")
```

PARA EL CALCULO Y CONCATENACIÓN DE LAS OPERACIONES SE UTILIZARON LOS SIGUIENTES MÉTODOS, PARA CONCATENAR LAS OPERACIONES SE UTILIZARON VARIOS FOR PARA IDENTIFICAR QUE TIPO DE OPERACIÓN, SI ERA SIMPLE O COMPLEJA.

```
def igual (self):
    for i in range (len(self.tupla)):
        llave=0
        key=0
        listaaux=[]
        text= ""
        text2="("
        if(len(self.tupla[i]))==2:
            text=str(self.tupla[i][0])+"("+str(self.tupla[i][1])+")"
            self.ordenados.append(text)
            if str(self.tupla[i][0])=="sen":
                self.tipo.append("Seno")
            if str(self.tupla[i][0])=="cos":
                self.tipo.append("Coseno")
            if str(self.tupla[i][0])=="tan":
                self.tipo.append("Tangente")
            if str(self.tupla[i][0])=="1/":
                self.tipo.append("Inverso")

        else:
            for p in range(len(self.signos)):
                if self.tupla[i][1]==self.signos[p]:
                    self.tipo.append("Compleja")
                    self.ordenados.append("-------")
                    llave=1
            if llave==0:

                listaaux.append(self.tupla[i][1])
                text+=(str(self.tupla[i][1]))
                var2=0
                listaaux.append(self.tupla[i][0])
                text+=(str(self.tupla[i][0]))
                #condición para saber si es una operación compleja
                if (len(self.tupla[i]))>3:
                    for j in range (2,len(self.tupla[i])):
                        for k in range (len(self.signos)):
```

```
                if (len(self.tupla[i]))>3:
                    for j in range (2,len(self.tupla[i])):
                        for k in range (len(self.signos)):
                            if self.tupla[i][j]==self.signos[k]:
                                for l in range (len(self.signos)):
                                    if self.tupla[i][j+2]==self.signos[l]:
                                        key =1
                                        var =j+2

                                        text2+=(str(self.tupla[i][j+1]))

                                        text2+=(str(self.tupla[i][j])+"(")
                                    if len(self.tupla[i])<6:

                                        var2=j
                            if key==1:
                                if j==var:

                                    text2+=(str(self.tupla[i][j+1]))
                                    text2+=(str(self.tupla[i][j]))
                                    text2+=(str(self.tupla[i][j+2]))
                            if j==var2:
                                text2+=(str(self.tupla[i][j+1]))
                                text2+=(str(self.tupla[i][j]))
                                text2+=(str(self.tupla[i][j+2]))
                                text2+=")"
                                text+=text2
                                self.ordenados.append(text)
                            if len(self.tupla[i])>5:
                                if j==len(self.tupla[i])-1:
                                    text2+=")"
                                    text+=text2
                                    self.ordenados.append(text)
                                    self.tipo.append("Compleja")
                else:
                    text+=(str(self.tupla[i][2]))
                    self.ordenados.append(text)
```

PARA CALCULAR LOS RESULTADOS SE UTILIZÓ UN MÉTODO EXTRA PARA SABER QUE SI SE TENÍA QUE SUMAR, RESTAR, MULTIPLICAR, ETC.

```
def total(self):
    for i in range(len(self.tupla)):
        llave= 0
        total = 0
        operando=0
        #saber el tamaño de la operación
        for s in range (len(self.signos)):
            if self.tupla[i][1]==self.signos[s]:
                llave=1
                if len(self.tupla[i])==7:

                    total=self.calcular(self.tupla[i][4],self.tupla[i][5],self.tupla[i][6])
                    operando=self.calcular(self.tupla[i][1],self.tupla[i][2],self.tupla[i][3])
                    total=self.calcular(self.tupla[i][0],operando,total)
                    self.totales.append(total)
                if len(self.tupla[i])==9:
                    total=self.calcular(self.tupla[i][7],self.tupla[i][8],None)
                    operando=self.calcular(self.tupla[i][5],self.tupla[i][6],None)
                    total=self.calcular(self.tupla[i][4],operando,total)
                    total=self.calcular(self.tupla[i][3],total,None)
                    operando=self.calcular(self.tupla[i][1],self.tupla[i][2],None)
                    total=self.calcular(self.tupla[i][0],operando,total)
                    self.totales.append(total)

            elif s==(len(self.signos))-1 and llave ==0:
                if len(self.tupla[i])==2:
                    total=self.calcular(self.tupla[i][0],self.tupla[i][1],None)
                    self.totales.append(total)
                elif len(self.tupla[i])>3 and len(self.tupla[i])<6:
                    total=self.calcular(self.tupla[i][2],self.tupla[i][3],self.tupla[i][4])
                    total=self.calcular(self.tupla[i][0],self.tupla[i][1],total)
                    self.totales.append(total)
                elif len(self.tupla[i])>5 and len(self.tupla[i])<8:
                    total=self.calcular(self.tupla[i][4],self.tupla[i][5],self.tupla[i][6])
                    total=self.calcular(self.tupla[i][2],self.tupla[i][3],total)
                    total=self.calcular(self.tupla[i][0],self.tupla[i][1],total)
                    self.totales.append(total)
```

```python
#Metodo para saber que signo es y operar
def calcular(self, signo, numero1, numero2):
    if signo =="+":
        calcular=float(numero1) + float(numero2)
        return calcular
    elif signo =="-":
        calcular=float(numero1) - float(numero2)
        return calcular
    elif signo =="/":
        calcular=float(numero1) / float(numero2)
        return calcular
    elif signo =="*":
        calcular=float(numero1) * float(numero2)
        return calcular
    elif signo =="**":
        calcular=float(numero2) ** float(numero1) #potencia
        return calcular
    elif signo =="sqrt":
        calcular = float(numero2)**(1/float(numero1))
        return calcular
    elif signo=="1/":
        calcular= 1/(numero1)
        return calcular
    elif signo =="%":
        calcular = float(numero1)%float(numero2)
        return calcular
    elif signo=="sen":
        calcular= math.sin(numero1)
        return calcular
    elif signo=="cos":
        calcular= math.cos(numero1)
        return calcular
    elif signo=="tan":
        calcular= math.sin(numero1)
        return calcular
```

LOS RESULTADOS SE MANDARON A UNA LISTA DE IGUAL FORMA QUE LA CONCATENACIÓN DE LAS OPERACIONES.

CLASES TOKEN:

CLASE CREADA PARA DEFINIR LAS PALABRAS RESERVADAS A UTILIZAR EN EL ANALIZADOR

Token.py > tokens

```python
from enum import Enum
import re
class tokens(Enum):
    TK_MENOR = "<"
    TK_E_NUMERO = "Numero"
    TK_MAYOR = ">"
    TK_NUMERO = "[0-9]*[.]?[0-9]*"
    TK_BARRAINV = "/"
    TK_E_OPERACION = "Operacion"
    TK_IGUAL = "="
    TK_OP_SUMA = "SUMA",
    TK_OP_MOD = "MOD",
    TK_OP_RESTA = "RESTA"
    TK_OP_MULTIPLICACION = "MULTIPLICACION"
    TK_E_TIPO = "Tipo"
    TK_OP_POTENCIA= "POTENCIA"
    TK_OP_RAIZ="RAIZ"
    TK_OP_INVERSO="INVERSO"
    TK_OP_SENO="SENO"
    TK_OP_COSENO="COSENO"
    TK_OP_TANGENTE="TANGENTE"
    TK_E_TEXTO="Texto"
    TK_E_PARRAFO="[a-zA-Z,À-ÿ\u00f1\u00d1,'+'-'-','*',0.0-9.0*,':','%','=','/','^','√','(',')']*"
    TK_E_Funcion="Funcion"
    TK_E_ESTILO="Estilo"
    TK_E_Titulo="Titulo"
    TK_DEMAS="(\s*[a-zA-ZÀ-ÿ\u00f1\u00d10-9\!\¡\#\$\%\&\(\)\\+\,\-\.\/\:\;\<\=\>\?\¿\[\\\]\^\_\{\|\}\~](\s*[a-zA-ZÀ-ÿ\u00f1\u0
    TK_MAYUS="[A-Z]*"
```

CLASE ERROR

```python
class error :
    def __init__(self, token , fila , columna, descripcion="") -> None:
        self.token= token
        self.fila=fila
        self.columna=columna
        self.descripcion=descripcion
    def getErrores(self):
        return {'token':self.token, 'fila':self.fila, 'columna':self.columna, 'descripcion':self.descripcion}
```

PARA GENERAR LOS REPORTES EN HTML, PRIMERO SE USÓ UN FOR PARA RECORRES LA LISTA DE COLORES Y TRADUCIRLOS AL INGLÉS PARA LA LECTURA DEL MISMO EN HTML, DESPUÉS SE GENERA EL CÓDIGO DE HTML EN UNA VARIABLE TIPO TEXTO, SE CONCATENAN LA CANTIDAD DE RESULTADOS QUE HAYAN EN LA LISTA USANDO UNA TABLA.

```
        pass
    def resultados(self,tipo,operacion,resultado,descripcion:str,titulo:str,colores,tamaños):
        coloresIngles=[]
        for i in range (len(colores)):

            if colores[i]=="AZUL":
                color="blue"
                coloresIngles.append(color)
            if colores[i]=="VERDE":
                color="green"
                coloresIngles.append(color)
            if colores[i]=="NEGRO":
                color="black"
                coloresIngles.append(color)
            if colores[i]=="ROJO":
                color="red"
                coloresIngles.append(color)
            if colores[i]=="AMARILLO":
                color="yellow"
                coloresIngles.append(color)
            if colores[i]=="NARANJA":
                color="orange"
                coloresIngles.append(color)
            if colores[i]=="MORADO":
                color="purple"
                coloresIngles.append(color)


        txt="""
        <html>
            <title>
            Resultados
            </title>
            <head>
            <link rel="icon" href="https://i.ibb.co/xhT1W0r/escudo10.png">
```

```
txt="""
<html>
    <title>
    Resultados
    </title>
    <head>
    <link rel="icon" href="https://i.ibb.co/xhT1W0r/escudo10.png">
    </head>
    <body style="background-color: #E2E5B8;">

    <font face="nunito,arial,verdana">

    <center>
    <FONT SIZE="""+str(tamaños[0])+""" COLOR="""+coloresIngles[0]+""" FACE="impact" style="text-align: center;">"""+ tit
    </center>
    <center>
    <FONT SIZE="""+str(tamaños[1])+""" COLOR="""+coloresIngles[1]+""" FACE="roman" style="text-align: center;">"""+ desc
    </center>
    <br><br/>
    <table style="height: 54px; width: 90%; border-collapse: collapse; margin-left: auto; margin-right: auto;" border="1
    <tbody>
    <tr style="height: 36px;background-color: #F67C62;">
    <td style="width: 25%; height: 36px; text-align: center;">No.</td>
    <td style="width: 25%; height: 36px; text-align: center;">Tipo</td>
    <td style="width: 25%; height: 36px; text-align: center;">Operación</td>
    <td style="width: 25%; height: 36px; text-align: center;">Resultado</td>
    </tr>
    """
for i in range(len(tipo)):
    txt+="""
        <tr style="height: 18px;background-color: #FFFFFF">
        <td style="width: 25%; height: 18px; text-align: center;">"""+str(i+1)+"""</td>
        <td style="width: 25%; height: 18px; text-align: center;">"""+str(tipo[i])+"""</td>
        <td style="width: 25%; height: 18px; text-align: center;">"""+str(operacion[i])+"""</td>
        <td style="width: 25%; height: 18px; text-align: center;">"""+str(resultado[i])+"""</td>
        </tr>
        """
```

POR ULTIMO SE GENERA EL ARCHIVO EN LA UBICACIÓN INDICADA.

```
Reporte=open("./Records/RESULTADOS_201902363.html","w+")
Reporte.write(txt)
Reporte.close
```

ESTE PROCESO SE REALIZÓ PARA EL HTML RESULTADOS Y HTML ERRORES.