# ESTRUCTURA DE PROGRAMA

# -\*- coding: cp1252 -\*-

#

# Objetivo: Operar dos números en formato ASCII y entregar su resultado

# en el mismo formato

# Autor: [Grupo 3] 10110-G-2

# Fecha: 24-10-2016

# Versión: 0.8

# BLOQUE DE DEFINICIÓN

# DEFINICIÓN DE CONSTANTES

# NUM0 = [[‘x’,’x’,’x’,’x’,’x’],…,[[‘x’,’x’,’x’,’x’,’x’]] representa NUM0

# Todos los números y signos siguen este mismo formato, y se decidió no

# incluirlo en el código por razones de orden

NUM0 = []

NUM1 = []

NUM2 = []

NUM3 = []

NUM4 = []

NUM5 = []

NUM6 = []

NUM7 = []

NUM8 = []

NUM9 = []

SMAS = []

SMEN = []

SMUL = []

SDIV = []

PUNT = []

ASCII = [NUM0,NUM1,NUM2,NUM3,NUM4,NUM5,NUM6,NUM7,NUM8,NUM9,SMAS,SMEN,SMUL,SDIV,PUNT]

STR = ['0','1','2','3','4','5','6','7','8','9','+','-','\*','/','.']

notFound = "NO se ha encontrado el archivo, por favor intente nuevamente"

notMatch = "el archivo no coincide con nada almacenado \n este error probablemente se deba a un caracter mal ubicado"

linea = "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"

O = 'OPERACIÓN\n'

R = 'RESULTADO\n'

fallo = 'Archivo incorrecto'

errorFila = 'El largo de alguna(s) fila(s) no corresponde(n) al requerido, modifique el archivo'

errorCaracter = "algun(os) caracter(es) no corresponde(n) a lo solicitado, modifique el archivo"

# IMPORTACIÓN DE FUNCIONES

from sys import \*

from Tkinter import \*

import tkFont

import tkMessageBox

from tkFileDialog import \*

# DEFINICIÓN DE FUNCIONES

# Función que crea una lista con cada línea del archivo de lectura

# Entrada: No hay

# Salida: Una lista de 7 elementos, cada elemento es una fila distinta

# del archivo de lectura

**def** leerArchivo(nombre)**:**

nombre = nombre.strip(".txt")

nombre = nombre + '.txt'

archivo = open(nombre,'r')

fila1 = []

fila2 = []

**for** i **in** range(7):

fila1.append(archivo.readline())

**if** i != 6:

fila2.append(fila1[i][0:len(fila1[i])-1])

**else**:

fila2.append(fila1[i][0:len(fila1[i])])

**if** (len(fila2[i]) + 1) % 6 != 0:

fila2 = True

return fila2

**for** i **in** range(1,6):

**if** len(fila2[i]) != len(fila2[i-1]):

fila2 = True

return fila2

**return** fila2

# Función que verifica que el archivo solo contenga ‘x’ y ‘.’

# Entrada: Lista de 7 elementos

# Salida: Un boolean si no hay errores, en caso contrario un mensaje

# en forma de string

**def** validacion**(**archivoEnUnaLinea**):**

check = True

error = False

fallo = 'Archivo incorrecto'

errorFila = 'El largo de alguna(s) fila(s) no corresponde(n) al requerido, modifique el archivo'

errorCaracter = "Uno o más caracteres no corresponden a lo solicitado, modifique el archivo"

mensaje = ""

**if** (archivoEnUnaLinea == True):

mensaje = ""

mensaje += "\n" + fallo + "\n" + errorFila

check = False

**return** mensaje

**else**:

**for** conjunto in archivoEnUnaLinea:

**for** letra in conjunto:

**if** **not**((letra == ".")or(letra == "x")):

error = True

**if** error:

mensaje = ""

mensaje += "\n" + fallo + "\n" + errorCaracter

**return** mensaje

**return** check

# Función que separa todos los caracteres de cada elemento de una lista

# Entrada: Una lista de 7 elementos

# Salida: Una lista de 7 elementos, cada elemento es una lista compuesta

# por cada carácter de la lista de entrada

**def** separarCaracter**(**archivoEnUnaLinea**):**

caracter = []

**for** i **in** range(7):

caracter.append([])

caracter[i] = list(archivoEnUnaLinea [i])

**return** caracter

# Función que agrupa en listas de a cinco elementos dentro de una lista # de 7 elementos y elimina el punto que separa a los números ASCII

# Entrada: Una lista compuesta por 7 listas

# Salida: Una lista que dentro de cada elemento hay varias listas, y que

# cada una de estas contiene 5 caracteres que están dentro de una misma

# fila en el archivo

**def** agruparCaracteres**(**caracter**):**

numero = []

**for** i **in** range(7):

numero.append([])

**for** j **in** range(int(len(caracter[i])/6)+1):

numero[i].append([])

**for** k **in** range(5):

numero[i][j].append(caracter[i][k+6\*j])

**return** numero

# Función que deja dentro de un mismo elemento todas las listas que

# representan a un número (o al operador) dentro de la operación

# matemática que se va a realizar

# Entrada: Una lista

# Salida: Una lista que sus elementos son números (u operadores) ASCII en

# el mismo formato que las constantes del código

**def** agruparNumeros**(**numero, caracter**):**

enteroMatriz = []

**for** i **in** range(int(len(caracter[0])/6)+1):

enteroMatriz.append([])

**for** j **in** range(7):

enteroMatriz[i].append(numero[j][i])

**return** enteroMatriz

# Función que reemplaza los números ASCII por números String

# Entrada: Una lista con números ASCII

# Salida: Una lista con números String

**def** reemplazar**(**enteroMatriz**):**

enteroNumero = []

**for** i **in** range(len(enteroMatriz)):

**for** j **in** range(len(ASCII)):

**if** enteroMatriz[i] == ASCII[j]:

enteroNumero.append(STR[j])

**return** enteroNumero

# Función que encuentra la posición del operador matemático

# Entrada: Una lista de String

# Salida: Un valor que representa la posición del operador en la lista de

# números String

**def** posicionOperador**(**enteroNumero**):**

indiceOperacion = None

**if** '+' **in** enteroNumero:

indiceOperacion = enteroNumero.index('+')

**elif** '-' **in** enteroNumero:

indiceOperacion = enteroNumero.index('-')

**elif** '\*' **in** enteroNumero:

indiceOperacion = enteroNumero.index('\*')

**else**:

indiceOperacion = enteroNumero.index('/')

**return** indiceOperacion

# Función que determina qué operación se va a realizar

# Entrada: Una lista de String

# Salida: Un string que puede ser ‘+’, ‘-‘, ‘\*’, o ‘/’

**def** operacion**(**enteroNumero**):**

tipoOperacion = '+'

**if** '+' **in** enteroNumero:

tipoOperacion = '+'

**elif** '-' **in** enteroNumero:

tipoOperacion = '-'

**elif** '\*' **in** enteroNumero:

tipoOperacion = '\*'

**elif**:

tipoOperacion = '/'

**return** tipoOperacion

# Función que convierte los números a enteros y luego los opera

# Entrada: Una lista de String, la posición del operador dentro de la

# lista, y el tipo de operación a realizar

# Salida: Un valor entero o decimal

**def** calculadora**(**enteroNumero, indiceOperacion, tipoOperacion**):**

numeroSt1 = ''

numeroSt2 = ''

error = False

resultadoOperacion = ""

primero = ""

segundo = ""

**if** indiceOperacion != None:

**for** i **in** range(indiceOperacion):

numeroSt1 = numeroSt1 + enteroNumero[i]

primero = int(numeroSt1)

**for** i **in** range(indiceOperacion+1, len(enteroNumero)):

numeroSt2 = numeroSt2 + enteroNumero[i]

segundo = int(numeroSt2)

**else**:

**for** i **in** range(len(enteroNumero)):

numeroSt1 = numeroSt1 + enteroNumero[i]

primero = int(numeroSt1)

segundo = 0

**if** tipoOperacion == '+':

resultadoOperacion = primero + segundo

**elif** tipoOperacion == '-':

resultadoOperacion = primero - segundo

**elif** tipoOperacion == '\*':

resultadoOperacion = primero \* segundo

**elif** tipoOperacion == '/':

**if** segundo != 0:

resultadoOperacion = round((float(primero) / float(segundo)),4)

**else**:

resultadoOperacion = "No se puede dividir por CERO"

**if** (resultadoOperacion != "No se puede dividir por CERO"):

**if** (resultadoOperacion - int(resultadoOperacion)) == 0:

resultadoOperacion = int(resultadoOperacion)

**return** resultadoOperacion

# Función que transforma el resultado a número ASCII

# Entrada: Una valor entero o decimal

# Salida: Una lista donde cada elemento es un número ASCII

**def** reconvertir**(**resultado**):**

listaAscii = []

resultadoString = list(str(resultado))

**for** i **in** range(len(resultadoString)):

**for** j **in** range(len(STR)):

**if** resultadoString[i] == STR[j]:

listaAscii.append(ASCII[j])

**return** listaAscii

# Función que transforma la lista de números ASCII al resultado final

# Entrada: Una lista

# Salida: Un string con el resultado de la operación en formato ASCII

**def** reordenar**(**resultadoAscii**):**

resultado1 = []

resultado2 = []

resultado3 = ''

**for** i **in** range(7):

resultado1.append([])

**for** j **in** range(len(resultadoAscii)):

resultado1[i].append(resultadoAscii[j][i])

**for** i **in** range(7):

resultado2.append('')

**for** j **in** range(len(resultado1[i])):

**for** k **in** range(5):

resultado2[i] = resultado2[i] + resultado1[i][j][k]

**if** j != len(resultado1[i]):

resultado2[i] = resultado2[i] + '.'

resultado2[i] = resultado2[i][0:len(resultado2[i])-1]

**for** i **in** range(7):

resultado3 = resultado3 + resultado2[i]

**if** i < 6:

resultado3 = resultado3 + '\n'

**return** resultado3

# BLOQUE PRINCIPAL

# El bloque principal se encuentra debidamente comentado dentro de la

# versión original del programa

**def** main**(**event**):**

#ENTRADA

**try**:

nombreArchivo = box.get()

archivoEnUnaLinea = leerArchivo(nombreArchivo)

**try**:

matrizInicial = archivoEnUnaLinea[0]+"\n"+archivoEnUnaLinea[1]+"\n"+archivoEnUnaLinea[2]+"\n"+archivoEnUnaLinea[3]+"\n"+archivoEnUnaLinea[4]+"\n"+archivoEnUnaLinea[5]+"\n"+archivoEnUnaLinea[6]+"\n"

**except** Exception **as** e:

matrizInicial = ""

#PROCESAMIENTO

**if** (validacion(archivoEnUnaLinea) == True):

caracter = separarCaracteres(archivoEnUnaLinea)

numero = agruparCaracteres(caracter)

enteroMatriz = agruparNumeros(numero,caracter)

enteroNumero = reemplazar(enteroMatriz)

indiceOperacion = posicionOperador(enteroNumero)

tipoOperacion = operacion(enteroNumero)

resultadoOperacion = calculadora(enteroNumero,indiceOperacion,tipoOperacion)

#SALIDA

**print** (linea+O+linea)

**print** matrizInicial

**print** (linea+R+linea)

numeroEnArreglo = reconvertir(resultadoOperacion)

matrizResultado = reordenar(numeroEnArreglo)

Operation.config(text=(linea+O+linea))

Matriz.config(text=matrizInicial)

Result.config(text=(linea+R+linea))

**if** (resultadoOperacion == "No se puede dividir por CERO"):

OUTPUT.config(text=resultadoOperacion)

**print** resultadoOperacion

**else**:

OUTPUT.config(text=matrizResultado)

**print** matrizResultado

**else**:

**if** (validacion(archivoEnUnaLinea) == ("\n" + fallo + "\n" + errorFila)):

tkMessageBox.showwarning("Warning", (validacion(archivoEnUnaLinea)))

Operation.config(text="")

Result.config(text="")

OUTPUT.config(text="")

Matriz.config(text="")

**elif** (validacion(archivoEnUnaLinea) == ("\n" + fallo + "\n" + errorCaracter)):

tkMessageBox.showwarning("Warning", (validacion(archivoEnUnaLinea)))

message.config(text="")

Operation.config(text="")

Result.config(text="")

OUTPUT.config(text="")

Matriz.config(text="")

**print** validacion(archivoEnUnaLinea)

**print** archivoEnUnaLinea

**except** SyntaxError:

**pass**

**except** NameError:

**pass**

**except** IOError:

**print** notFound

tkMessageBox.showerror("Error", notFound)

message.config(text="")

Operation.config(text="")

Result.config(text="")

OUTPUT.config(text="")

Matriz.config(text="")

**except** IndexError:

**print** notMatch

tkMessageBox.showwarning("Warning", fallo+"\n"+notMatch)

message.config(text="")

Operation.config(text="")

Result.config(text="")

OUTPUT.config(text="")

Matriz.config(text="")