```
import Pila

"""
Funcion que es invocada para crear Tokens dada una cadena de texto. La funciono ordena una lista en funcion del tipo de caracter qu
"""
def getTokens(cadena):
  length=len(cadena)
  i=0
  lista = []
  guardar=""
  while i<length:
    car = cadena[i]
    dato = tipoDato(car)
    # caso de un caracter distinto de numero o de "-" o de "."
    if dato == "caracter" or dato == "parIzq" or dato == "parDer":
      if guardar!="":
        lista.append(guardar)
      lista.append(car)
      guardar=""
    # caso especial para resta
    elif dato == "masMenos":
      if i == 0 or (tipoDato(cadena[i-1]) != "numero" and tipoDato(cadena[i-1]) != "parDer"):
        guardar+=car
      else:
        if guardar!="":
          lista.append(guardar)
        lista.append(car)
        guardar=""
    elif dato=="punto" or dato=="numero":
      guardar+=car
    if i==len(cadena)-1 and guardar!="":
      lista.append(guardar)
    i+=1
  return lista

"""
Funcion que recibe un caracter como parametro y es convertido a su valor en codigo ascii. Su funcion es devolver la función que des
"""
def tipoDato(caracter):
  valor = ord(caracter)
  if valor == 46:
```

```
    return "punto"
  elif valor == 45 or valor == 43:
    return "masMenos"
  elif valor == 40:
    return "parIzq"
  elif valor == 41:
    return "parDer"
  elif valor < 48 or valor > 57:
    return "caracter"
  else:
    return "numero"

"""
Funcion que revisa un token donde debe existir
a lo mas un punto por sub-arreglo.
"""
def puntoBien(token):
  length=len(token)
  i=0
  cont=0
  while i<length and cont < 2:
    if token[i]==".":
      cont+=1
    i+=1
  return cont < 2
"""
En esta función se verifica que tanto los tokens generados sean válidos, como que el orden de los operadores y operandos sea el cor
"""
def valida(expresion): #expresion es una lista? creo que si
  i=0
  length=len(expresion)
  todoBien = True
  pila = Pila.Pila()
  mathError = False
  while i<length and todoBien:
    elem=expresion[i]

    if puntoBien(elem):
      if i!=length-1 and elem=="/" and expresion[i+1]=="0":
        todoBien=False
        mathError=True
```

```
      if len(elem) == 1 and tipoDato(elem)=="masMenos" and (i == length-1 or expresion[i+1]=="*" or expresion[i+1]=="/"):
        todoBien = False
      if elem=="(":
        pila.push(elem)
        if i==len(expresion)-1:
          todoBien=False
        elif expresion[i+1]==")":
          todoBien=False
        # caracter : (*, (/, (), (a
        elif len(expresion[i+1]) == 1 and (tipoDato(expresion[i+1])=="caracter" or tipoDato(expresion[i+1])=="parDer"):
          todoBien = False
      elif elem == ")":
        try:
          pila.extrae()
        except:
          todoBien = False
      if i!=length-1 and len(elem)==1 and tipoDato(elem)=="caracter" and len(expresion[i+1])==1 and tipoDato(expresion[i+1])=="cara
        todoBien = False
      if i==length-1 and len(elem)==1 and tipoDato(elem)=="caracter":
        todoBien = False
    else:
      todoBien = False
    i+=1
  if not pila.estaVacia() or length == 0:
    todoBien = False
  if not todoBien and not mathError: #no todoBien y no hay mathError
    tupla=(todoBien,"Syntax ERROR")
  elif not todoBien: #todoBien==False
    tupla=(todoBien, "Math ERROR")
  else:
    tupla=(todoBien,"")
  return tupla
"""
Funcion que recibe un token y limpia los signos de - extras en la expresion
"""
def balanceaSigno(token):
  if len(token)>1:
    guardar=""
    contador = 0
    for char in token:
      if char == "-":
```

```
        contador+=1
      elif char != "+":
        guardar+=char
    if contador%2!=0:
      guardar = "-" + guardar
    token=guardar
  return token
"""
Función que maneja un arreglo de tokens, donde a cada uno le añade un signo o los balancea para su próxima operación
"""
def limpiaTokens(lista):
  i=0
  while i<len(lista):
    lista[i]=balanceaSigno(lista[i])
    if i!= len(lista)-1 and lista[i]==")" and (lista[i+1]=="(" or len(lista[i+1])>1 or tipoDato(lista[i+1])=="numero"):
      lista=añadeSigno(lista,i+1)
    elif i!=0 and lista[i]=="(" and (len(lista[i-1])>1 or tipoDato(lista[i-1])=="numero"):
      lista=añadeSigno(lista,i)
    i+=1
  return lista

"""
Funcion que añade el signo de multiplicacion a un arreglo de caracteres
dado el caso de  3(5), por ejemplo.
"""
def añadeSigno(arr,empieza):
  aux=[]
  i=empieza
  j=0

  #copio el arreglo
  while i<len(arr):
    aux.append(arr[i])
    i+=1

  arr[empieza]="*"
  i=empieza+1
  while j<len(aux)-1:
    arr[i]=aux[j]
    j+=1
    i+=1
```

```
  arr.append(aux[j])
  return arr

import Node
"""
Recibe la lista de getTokens, se crea un arbol a partir de una expresion
ya validada y convertida en Token. Se declaran dos pilas en las que se guardan operandos y operadores
"""
def creaArbol(lista):
  tuplaVal = valida(lista)
  if len(lista) != 0 and tuplaVal[0]:
    lista=limpiaTokens(lista)
    #pilaA es la pila de operadores
    pilaA=Pila.Pila()
    #pilaB es la pila de Arboles
    pilaB=Pila.Pila()
    for token in lista:
      #caso en el que es un número
      if len(token)>1 or tipoDato(token)=="numero":
        nodo=Node.Node(token)
        pilaB.push(nodo)
      elif token=="(":
        pilaA.push(token)
      elif token==")":
        # por el valida nunca truena
        while pilaA.peek()!="(":
          op=pilaA.extrae()
          p=pilaB.extrae()
          s=pilaB.extrae()
          nuevo=Node.Node(op)
          nuevo.setDer(p)
          nuevo.setIzq(s)
          pilaB.push(nuevo)
        pilaA.extrae()
      #caso en el que es un operador
      else:
        while not pilaA.estaVacia() and pilaA.peek()!="(" and prioridad(token,pilaA.peek())!="mayor":
          nuevo=Node.Node(pilaA.extrae())
          p=pilaB.extrae()
          s=pilaB.extrae()
          nuevo.setDer(p)
```

```
          nuevo.setIzq(s)
          pilaB.push(nuevo)
        pilaA.push(token)

    while not pilaA.estaVacia():
      nuevo=Node.Node(pilaA.extrae())
      p=pilaB.extrae()
      s=pilaB.extrae()
      nuevo.setDer(p)
      nuevo.setIzq(s)
      pilaB.push(nuevo)
    return pilaB.extrae()
  else:
    return tuplaVal[1]

"""
Funcion que recibe como parametro signos de operación
la cual determina la prioridad de operación.
"""
def prioridad(uno,dos):
  if uno=="+" or uno=="-":
    if dos=="+" or dos=="-":
      return "igual"
    else:
      return "menor"
  else:
    if dos=="+" or dos=="-":
      return "mayor"
    else:
      return "igual"

"""Funcion para convertir a flotante"""
def convierte(numero):
  numero = float(numero)
  return numero

"""Funcion de evaluar (resume las tres etapas en una)"""
def evalua(expresion):
  tokens=getTokens(expresion)
  #print(tokens)
  nodo=creaArbol(tokens)
```

```
  #print(tokens)
  """if nodo!=None:
    print(Node.postOrden(nodo))
  """
  if type(nodo) == str:
    return nodo
  else:
    return evaluaRec(nodo)
"""
verifica que el nodo recibido no sea nulo, luego realiza un recorrido en postorden y conforme va sacando izquierdos y derechos se p
"""

def evaluaRec(nodo):
  if nodo != None :
    dato=nodo.getDato()
    if len(dato)>1 or tipoDato(dato)=="numero":
      dato=convierte(dato)
      return dato
    #caso de ser operador que jamás va a estar hasta abajo
    else:
      izq = evaluaRec(nodo.getIzq())
      der = evaluaRec(nodo.getDer())
      if der == "Math ERROR" or izq == "Math ERROR":
        return "Math ERROR"
      if dato == "+":
        return izq+der
      elif dato == "-":
        return izq-der
      elif dato == "*":
        return izq*der
      else:
        if der == 0:
          return "Math ERROR"
        else:
          return izq/der
  return ""
```