*#dada una lista enlazada, imprimala en orden inverso sin agregarle mas*

*#punteros o nodos centinela*

*#en primer lugar creamos una lista enlazada simple, la de siempre.*

*#el nodo*

**class** nodo(object):

**def** \_\_init\_\_(self, value = 0, next = **None**):

self.value = value

self.next = next

*#la lista*

**class** listaEnlazada(object):

**def** \_\_init\_\_(self, head = **None**):

self.head = head

**def** insertar(self, value):

**if**(self.head == **None**):

self.head = nodo(value, **None**)

**return**

aux = self.head *#creamos nuestro nodo auxiliar para recorrer la lista*

**while**(aux != **None**):

*#iteramos nuestra lista hasta llegar al ultimo nodo de esta*

*#una vez alli insertamos nuestro valor al final de la lista*

**if**(aux.next == **None**):

aux.next = nodo(value, **None**)

**return**

aux = aux.next

**def** imprimir(self):*#imprimir desde cabeza al final, #esto demora O(n)*

**if**(self.head == **None**):

**return***#si lista esta vacia no hay nada que imprimir*

**else**:

aux = self.head *#en caso contrario la recorremos e imprimimos*

**while**(aux != **None**):

**print** (aux.value,) *#la coma al final es para imprimir sin saltos*

aux = aux.next

**print**('')

"""Ahora para imprimir en reversa no es tan sencillo como el insertar

e imprimir, pero tampoco es algo de otro mundo, en primer lugar dibujar

la lista y descubrir la logica que hay para imprimir en reversa, sin usar

mas nodos centinela ni agregando un puntero en direccion contraria"""

**def** lista\_size(self):*#esto demora O(n)*

**if**(self.head == **None**):

**return**

aux = self.head

cont = 0

**while**(aux != **None**):

cont += 1

aux = aux.next

**return** cont

**def** imprimir\_reversa(self):

**if**(self.head == **None**):*#como de costumbre, siempre checkear si esta vacia*

**return**

N = self.lista\_size()

**while**(N >= 1):*#toma O(n^2), se itera n veces algo que itera n veces*

aux = self.head*#eso implica que demora n\*n = n^2*

cont = 1

**while**(aux != **None**):

**if**(cont == N):

**print** (aux.value,)

cont = 1

**break**

cont += 1

aux = aux.next

N -= 1

*#creamos la lista*

l = listaEnlazada(**None**)

*#le insertamos datos*

**for** i **in** range(0, 10):

l.insertar(i)

*#imprimimos en orden(de cabeza a final)*

l.imprimir()

*#imprimimos en reversa(final a cabeza)*

l.imprimir\_reversa()

**class** Node(object):

**def** \_\_init\_\_(self, \_value):

self.next = **None**

self.value = \_value

**class** Queue(object):

**def** \_\_init\_\_(self):

self.primero = **None**

self.ultimo = **None**

**def** isEmpty(self):

**if** self.ultimo == **None**:

**return** True

**else**:

**return** False

**def** Enqueue(self, node):

**if** self.isEmpty():

self.primero = self.ultimo = node

**else**:

self.ultimo.next = node

self.ultimo = node

**def** Dequeue(self):

**if** self.isEmpty():

**return** -1

**elif** self.primero == self.ultimo:

value = self.primero.value

self.primero = self.ultimo = **None**

**return** value

**else**:

value = self.primero.value

self.primero = self.primero.next

**return** value

**class** ColaUrgencia(object):

**def** \_\_init\_\_(self):

self.ColaPac1 = Queue()

self.ColaPac2 = Queue()

**def** encolarUrgencia(self, tipoPaciente, nombrePaciente):

paciente = Node(nombrePaciente)

**if** tipoPaciente == 3:

**print**("Enviando al hospital al paciente: " + paciente.value)

**elif** tipoPaciente == 2:

self.ColaPac2.Enqueue(paciente)

**print**("Encolando en cola 2 a paciente " + paciente.value);

**elif** tipoPaciente == 1:

self.ColaPac1.Enqueue(paciente)

**print**("Encolando en cola 1 a paciente " + paciente.value);

**def** desencolarUrgencia(self):

**if** self.estaVacioUrgencia() == False:

**if** self.ColaPac2.isEmpty() == False:

**return** "Desencolando de cola 2: " + self.ColaPac2.Dequeue()

**else**:

**return** "Desencolando de cola 1: " + self.ColaPac1.Dequeue()

**else**:

**return** "No hay nadie quien desencolar"

**def** estaVacioUrgencia(self):

**if** self.ColaPac1.isEmpty() **and** self.ColaPac2.isEmpty():

**return** True

**else**:

**return** False

ColaUrg = ColaUrgencia()

ColaUrg.encolarUrgencia(3, "Martin")

ColaUrg.encolarUrgencia(1, "Juan")

ColaUrg.encolarUrgencia(1, "Diego")

ColaUrg.encolarUrgencia(2, "Pedro")

ColaUrg.encolarUrgencia(1, "Pollo")

ColaUrg.encolarUrgencia(1, "Stop")

**print**(ColaUrg.desencolarUrgencia())

**print**(ColaUrg.desencolarUrgencia())

ColaUrg.encolarUrgencia(2, "Brian")

**print**(ColaUrg.desencolarUrgencia())

**print**(ColaUrg.desencolarUrgencia())

**print**(ColaUrg.desencolarUrgencia())

**print**(ColaUrg.desencolarUrgencia())

**print**(ColaUrg.desencolarUrgencia())

**print**(ColaUrg.desencolarUrgencia())

**print**(ColaUrg.desencolarUrgencia())

**print**(ColaUrg.desencolarUrgencia())

**print**(ColaUrg.desencolarUrgencia())

**class** Node(object):

**def** \_\_init\_\_(self, \_value):

self.next = **None**

self.value = \_value

**class** Stack(object):

**def** \_\_init\_\_(self):

self.top=**None**

**def** isEmpty(self):

**if** self.top == **None**:

**return** True

**else**:

**return** False

**def** Push(self, node):

**if** self.isEmpty():

self.top=node

**else**:

node.next=self.top

self.top=node

**def** Pop(self):

**if** self.isEmpty():

**return** -1

**elif** self.top.next==**None**:

value=self.top.value

self.top=**None**

**return** value

**else**:

value=self.top.value

self.top=self.top.next

**return** value

**def** ParentesisSolver(Parentesis):

Pila=Stack()

**for** i **in** Parentesis:

curNode=Node(i)

**if** i=='(':

Pila.Push(curNode)

**elif** i==')':

value=Pila.Pop()

**if** value == -1:

**return** False

**if** Pila.isEmpty():

**return** True

**else**:

**return** False

case1 = "())(()"

case2 = "(())()"

case3 = "((((())()()))()())"

case4 = "((((())()()))()())("

case5 = ")((((())()()))()())("

**print**(ParentesisSolver(case1))

**print**(ParentesisSolver(case2))

**print**(ParentesisSolver(case3))

**print**(ParentesisSolver(case4))

**print**(ParentesisSolver(case5))

**def** mergeSort(alist):

**if** len(alist)>1:

mid = len(alist)//2

lefthalf = alist[:mid]

righthalf = alist[mid:]

mergeSort(lefthalf)

mergeSort(righthalf)

i=0

j=0

k=0

**while** i<len(lefthalf) **and** j<len(righthalf):

**if** lefthalf[i]<righthalf[j]:

alist[k] = lefthalf[i]

i+=1

**else**:

alist[k] = righthalf[j]

j+=1

k+=1

**while** i<len(lefthalf):

alist[k] = lefthalf[i]

i+=1

k+=1

**while** j<len(righthalf):

alist[k] = righthalf[j]

j+=1

k+=1

list2 = [2,1,3,10,40,60,100,60,24,3,5,6]

mergeSort(list2)

**print**(list2)

**class** nodo(object):

**def** \_\_init\_\_(self, \_value):

self.pSig=**None**

self.pAnt=**None**

self.value=\_value

**def** getElemento(self):

**return** self.value

**class** listaDoble(object):

**def** \_\_init\_\_(self):

self.primero=**None**

self.ultimo=**None**

**def** getVacio(self):

**if** self.primero==**None**:

**return** True

**def** printPrimeroUltimo(self):

**if** self.getVacio()==True:

**print** ('Esta vacia la lista')

**else**:

validar=True

temp=self.primero

**while**(validar):

**print**(temp.getElemento())

**if** temp==self.ultimo:

validar=False

**else**:

temp=temp.pSig

**def** printUltimoPrimero(self):

**if** self.getVacio()==True:

**print**('Esta vacia la lista')

**else**:

validar=True

temp=self.ultimo

**while** (validar):

**print**(temp.getValor)

**if** temp==self.primero:

validar=False

**else**:

temp=temp.pAnt

**def** EliminarPrimero(self):

**if** self.getVacio()==True:

**print** ('La lista ya esta vacia')

**elif** self.primero==self.ultimo:

self.primero=self.ultimo=**None**

**print** ('La lista esta vacia')

**else**:

temp=self.primero

self.primero=temp.pSig

self.primero.pAnt=**None**

temp=**None**