Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

LABORATORIO INTRODUCCION A LA PROGRAMACION Y COMPUTACION 2 Sección D

OSCAR ROBERTO VELÁSQUEZ LEÓN

Practica auxiliar

Nombre	Carné	Sección
Brandon Orlando Seijas Morales	202010035	D

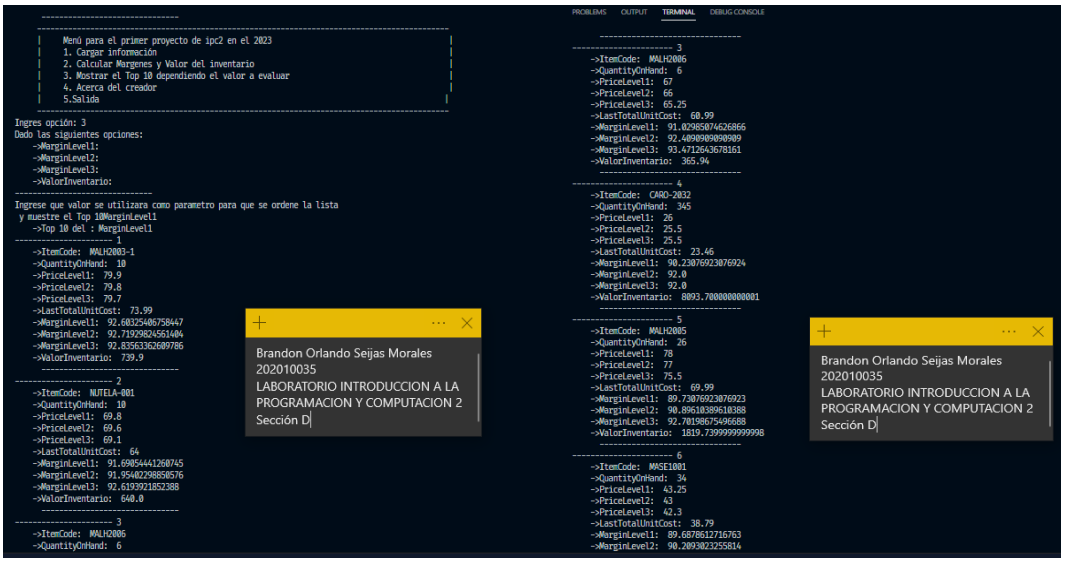
Link del repositorio

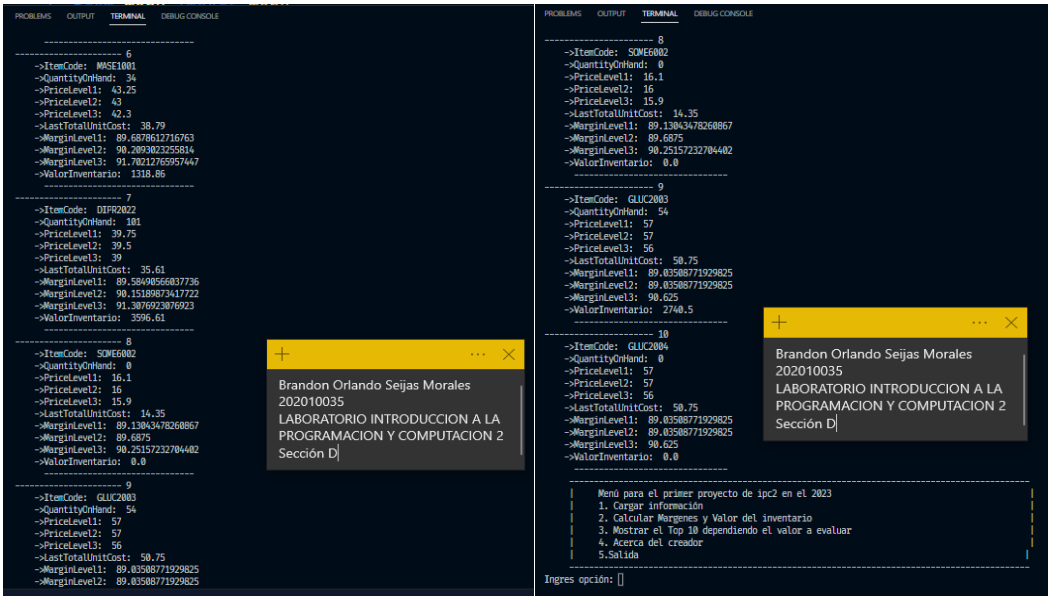
https://github.com/brandonseijas/IPC2_PracticaAuxiliar_2023.git

Top 10

Pantallazos de los resultados de los top 10 de los 3 niveles con mayor rentabilidad y el top 10 de productos con mayor valor del inventario

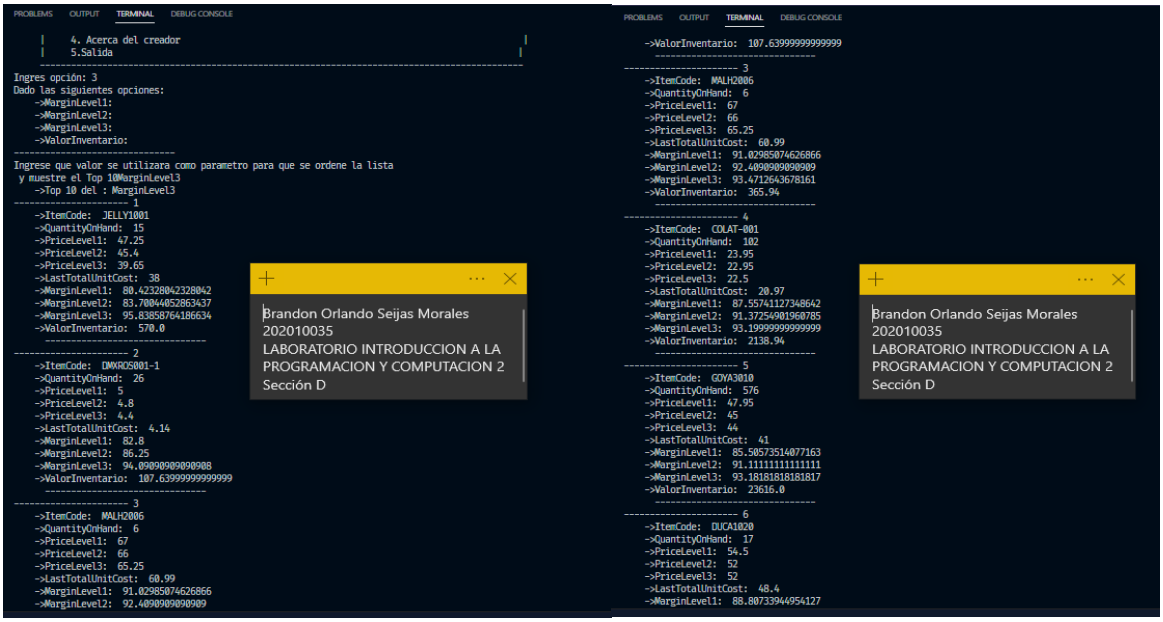
MarginLevel1

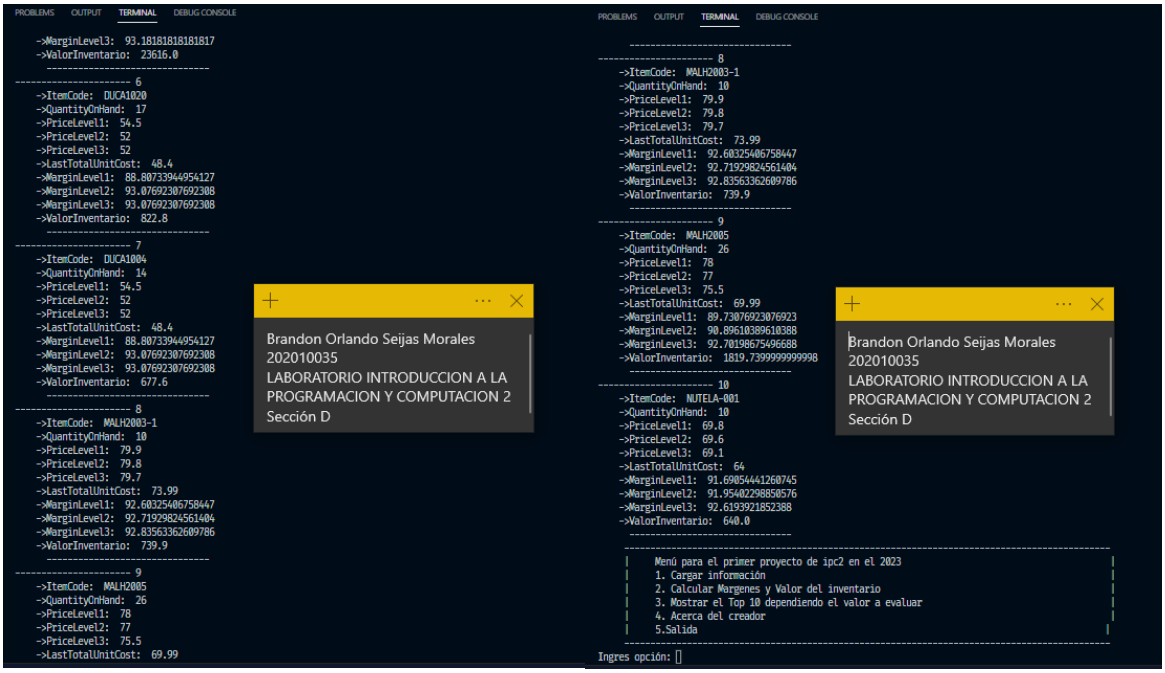




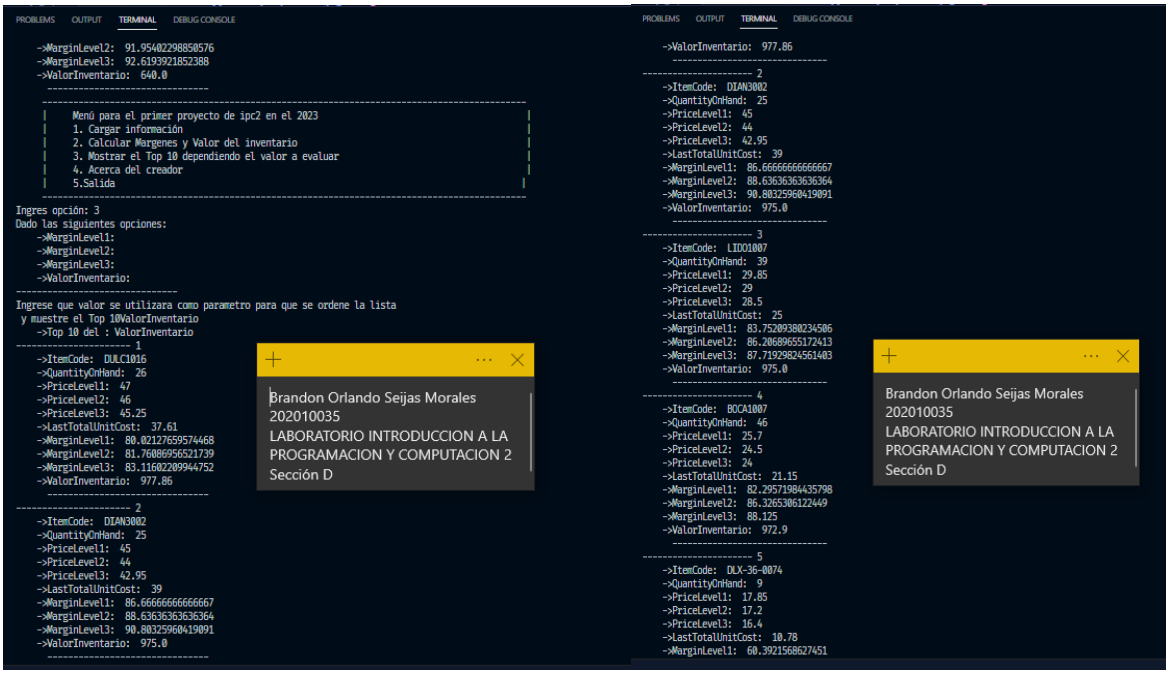
MarginLevel2

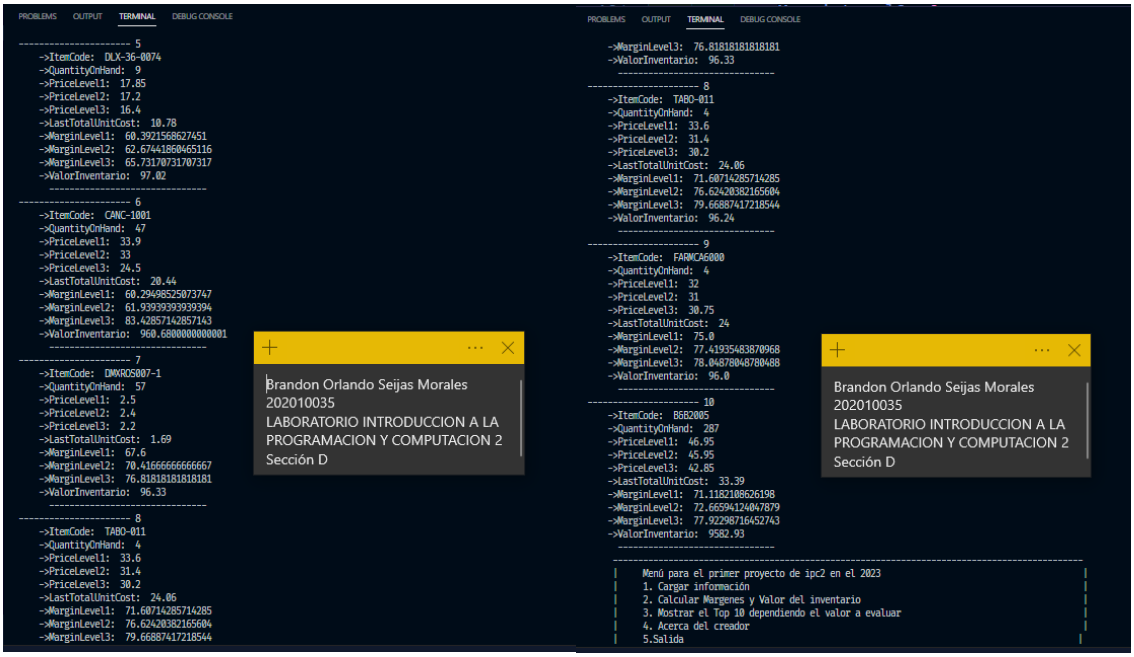
MarginLevel3





ValorInventario





- Código de la lista simplemente enlazada

```
class nodoempresa():

   def __init__(self):

      self.ItemCode = ""

      self.QuantityOnHand = ""

      self.PriceLevel1 = ""

      self.PriceLevel2 = ""

      self.PriceLevel3 = ""

      self.LastTotalUnitCost = ""

      self.siguiente = None

      self.antes = None

class Lista:
   def __init__(self):
      self.primero = None
      self.ultimo = None
      self.size = 0

   def vacia(self):
      return self.primero == self.ultimo == None

   def agregarNodo(self, nodoempresa):
      nuevamuestra = nodoempresa
      if self.vacia():
         self.primero = self.ultimo = nuevamuestra
      elif self.primero == self.ultimo:
         self.ultimo = nuevamuestra
         self.primero.siguiente = self.ultimo
         self.ultimo.antes = self.primero
      else:
         self.ultimo.siguiente = nuevamuestra
         nuevamuestra.antes = self.ultimo
         self.ultimo = nuevamuestra
      self.size = self.size +1


   def mostrardato(self):
      actual = self.primero
      while (actual != None):
         print(f"    ->ItemCode: ",str(actual.ItemCode))
```

```
        print(f"    ->QuantityOnHand: ",str(actual.QuantityOnHand))
        print(f"    ->PriceLevel1: ", str(actual.PriceLevel1))
        print(f"    ->PriceLevel2: ",str(actual.PriceLevel2))
        print(f"    ->PriceLevel3: ",str(actual.PriceLevel3))
        print(f"    ->LastTotalUnitCost: ", str(actual.LastTotalUnitCost))
        print("     ------------------------------")
        actual = actual.siguiente
```

```
class nodoempresa():
    def __init__(self):
        self.ItemCode = ""
        self.QuantityOnHand = ""
        self.PriceLevel1 = ""
        self.PriceLevel2 = ""
        self.PriceLevel3 = ""
        self.LastTotalUnitCost = ""
        self.siguiente = None
        self.antes = None
class Lista:
    def __init__(self):
        self.primero = None
        self.ultimo = None
        self.size = 0

    def vacia(self):
        return self.primero == self.ultimo == None

    def agregarNodo(self, nodoempresa):
        nuevamuestra = nodoempresa
        if self.vacia():
            self.primero = self.ultimo = nuevamuestra
        elif self.primero == self.ultimo:
            self.ultimo = nuevamuestra
            self.primero.siguiente = self.ultimo
            self.ultimo.antes = self.primero
        else:
            self.ultimo.siguiente = nuevamuestra
            nuevamuestra.antes = self.ultimo
            self.ultimo = nuevamuestra
        self.size = self.size +1


    def mostrardato(self):
        actual = self.primero
        while (actual != None):
            print(f"    ->ItemCode: ",str(actual.ItemCode))
            print(f"    ->QuantityOnHand: ",str(actual.QuantityOnHand))
            print(f"    ->PriceLevel1: ", str(actual.PriceLevel1))
            print(f"    ->PriceLevel2: ",str(actual.PriceLevel2))
            print(f"    ->PriceLevel3: ",str(actual.PriceLevel3))
            print(f"    ->LastTotalUnitCost: ", str(actual.LastTotalUnitCost))
            print("     ------------------------------")
            actual = actual.siguiente
```

- Código para ordenar el arreglo

```
class Nodo:

  def __init__(self, data):

    self.data = data

    self.next = None
```

```
        self.prev = None
class Datos:
    def __init__(self, ItemCode, QuantityOnHand, PriceLevel1, PriceLevel2, PriceLevel3,
LastTotalUnitCost, MarginLevel1,MarginLevel2,MarginLevel3,ValorInventario):
        self.ItemCode = ItemCode
        self.QuantityOnHand = QuantityOnHand
        self.PriceLevel1 = PriceLevel1
        self.PriceLevel2 = PriceLevel2
        self.PriceLevel3 = PriceLevel3
        self.LastTotalUnitCost = LastTotalUnitCost
        self.MarginLevel1 = MarginLevel1
        self.MarginLevel2 = MarginLevel2
        self.MarginLevel3 = MarginLevel3
        self.ValorInventario = ValorInventario
class ListaDoble:
    def __init__(self):
        self.head = None

    def agregar(self, data):
        nuevo_nodo = Nodo(data)
        if self.head is None:
            self.head = nuevo_nodo
        else:
            actual = self.head
            while actual.next is not None:
                actual = actual.next
            actual.next = nuevo_nodo
            nuevo_nodo.prev = actual
```

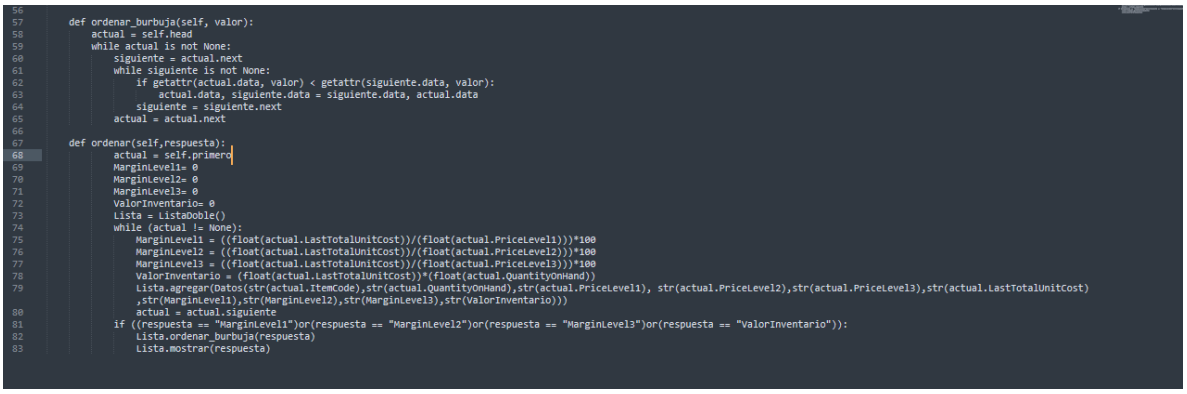
```
def mostrar(self,respuesta):
    i=0
    if self.head is None:
        print("La lista está vacía")
    else:
        actual = self.head
        print(f"    ->Top 10 del : {respuesta}")
        while actual is not None:
            if (i<10):
                print(f"---------------------- {i+1}")
                print(f"    ->ItemCode: ",str(actual.data.ItemCode))
                print(f"    ->QuantityOnHand: ",str(actual.data.QuantityOnHand))
                print(f"    ->PriceLevel1: ", str(actual.data.PriceLevel1))
                print(f"    ->PriceLevel2: ",str(actual.data.PriceLevel2))
                print(f"    ->PriceLevel3: ",str(actual.data.PriceLevel3))
                print(f"    ->LastTotalUnitCost: ", str(actual.data.LastTotalUnitCost))
                print(f"    ->MarginLevel1: ", str(actual.data.MarginLevel1))
                print(f"    ->MarginLevel2: ", str(actual.data.MarginLevel2))
                print(f"    ->MarginLevel3: ", str(actual.data.MarginLevel3))
                print(f"    ->ValorInventario: ", str(actual.data.ValorInventario))
                print("     ------------------------------")
                i = i+1
            actual = actual.next


def ordenar_burbuja(self, valor):
    actual = self.head
    while actual is not None:
        siguiente = actual.next
        while siguiente is not None:
```

```
            if getattr(actual.data, valor) < getattr(siguiente.data, valor):
                actual.data, siguiente.data = siguiente.data, actual.data
            siguiente = siguiente.next
        actual = actual.next


    def ordenar(self,respuesta):
        actual = self.primero
        MarginLevel1= 0
        MarginLevel2= 0
        MarginLevel3= 0
        ValorInventario= 0
        Lista = ListaDoble()
        while (actual != None):
            MarginLevel1 = ((float(actual.LastTotalUnitCost))/(float(actual.PriceLevel1)))*100
            MarginLevel2 = ((float(actual.LastTotalUnitCost))/(float(actual.PriceLevel2)))*100
            MarginLevel3 = ((float(actual.LastTotalUnitCost))/(float(actual.PriceLevel3)))*100
            ValorInventario = (float(actual.LastTotalUnitCost))*(float(actual.QuantityOnHand))

Lista.agregar(Datos(str(actual.ItemCode),str(actual.QuantityOnHand),str(actual.PriceLevel1),
str(actual.PriceLevel2),str(actual.PriceLevel3),str(actual.LastTotalUnitCost)
,str(MarginLevel1),str(MarginLevel2),str(MarginLevel3),str(ValorInventario)))
            actual = actual.siguiente
        if ((respuesta == "MarginLevel1")or(respuesta == "MarginLevel2")or(respuesta ==
"MarginLevel3")or(respuesta == "ValorInventario")):
            Lista.ordenar_burbuja(respuesta)
            Lista.mostrar(respuesta)
```



```
    def ordenar_burbuja(self, valor):
        actual = self.head
        while actual is not None:
            siguiente = actual.next
            while siguiente is not None:
                if getattr(actual.data, valor) < getattr(siguiente.data, valor):
                    actual.data, siguiente.data = siguiente.data, actual.data
                siguiente = siguiente.next
            actual = actual.next

    def ordenar(self,respuesta):
            actual = self.primero
            MarginLevel1= 0
            MarginLevel2= 0
            MarginLevel3= 0
            ValorInventario= 0
            Lista = ListaDoble()
            while (actual != None):
                MarginLevel1 = ((float(actual.LastTotalUnitCost))/(float(actual.PriceLevel1)))*100
                MarginLevel2 = ((float(actual.LastTotalUnitCost))/(float(actual.PriceLevel2)))*100
                MarginLevel3 = ((float(actual.LastTotalUnitCost))/(float(actual.PriceLevel3)))*100
                ValorInventario = (float(actual.LastTotalUnitCost))*(float(actual.QuantityOnHand))
                Lista.agregar(Datos(str(actual.ItemCode),str(actual.QuantityOnHand),str(actual.PriceLevel1), str(actual.PriceLevel2),str(actual.PriceLevel3),str(actual.LastTotalUnitCost)
                ,str(MarginLevel1),str(MarginLevel2),str(MarginLevel3),str(ValorInventario)))
                actual = actual.siguiente
            if ((respuesta == "MarginLevel1")or(respuesta == "MarginLevel2")or(respuesta == "MarginLevel3")or(respuesta == "ValorInventario")):
                Lista.ordenar_burbuja(respuesta)
                Lista.mostrar(respuesta)
```

Código para ordenar el arreglo depende de una llave se puede ordenar y obtener el top 10 dependiendo si elige cual de los 3 niveles con mayor rentabilidad o los productos con mayor valor de inventario desea ordenar.

- Código para realizar los cálculos de los márgenes

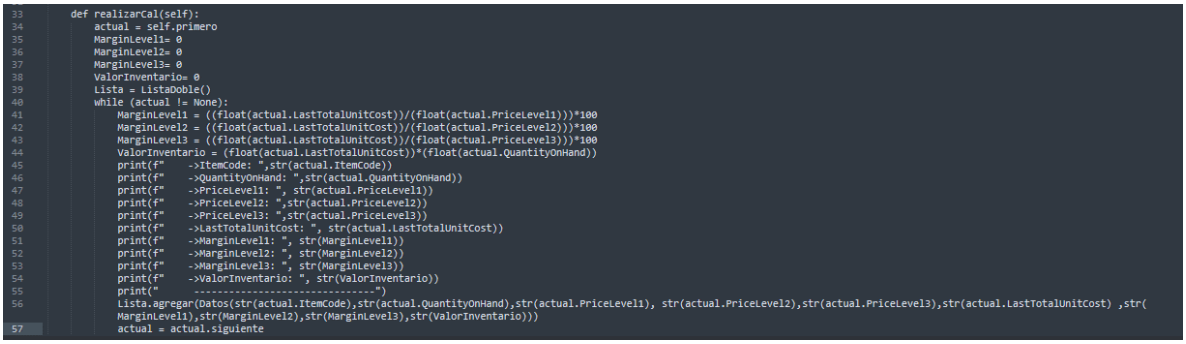
```
class Nodo:
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None
class Datos:
    def __init__(self, ItemCode, QuantityOnHand, PriceLevel1, PriceLevel2, PriceLevel3,
LastTotalUnitCost, MarginLevel1,MarginLevel2,MarginLevel3,ValorInventario):
        self.ItemCode = ItemCode
        self.QuantityOnHand = QuantityOnHand
        self.PriceLevel1 = PriceLevel1
        self.PriceLevel2 = PriceLevel2
        self.PriceLevel3 = PriceLevel3
        self.LastTotalUnitCost = LastTotalUnitCost
        self.MarginLevel1 = MarginLevel1
        self.MarginLevel2 = MarginLevel2
        self.MarginLevel3 = MarginLevel3
        self.ValorInventario = ValorInventario
class ListaDoble:
    def __init__(self):
        self.head = None

    def agregar(self, data):
        nuevo_nodo = Nodo(data)
        if self.head is None:
            self.head = nuevo_nodo
        else:
            actual = self.head
            while actual.next is not None:
                actual = actual.next
```

```
            actual.next = nuevo_nodo
            nuevo_nodo.prev = actual

    def realizarCal(self):
        actual = self.primero
        MarginLevel1= 0
        MarginLevel2= 0
        MarginLevel3= 0
        ValorInventario= 0
        Lista = ListaDoble()
        while (actual != None):
            MarginLevel1 = ((float(actual.LastTotalUnitCost))/(float(actual.PriceLevel1)))*100
            MarginLevel2 = ((float(actual.LastTotalUnitCost))/(float(actual.PriceLevel2)))*100
            MarginLevel3 = ((float(actual.LastTotalUnitCost))/(float(actual.PriceLevel3)))*100
            ValorInventario =
(float(actual.LastTotalUnitCost))*(float(actual.QuantityOnHand))
            print(f"    ->ItemCode: ",str(actual.ItemCode))
            print(f"    ->QuantityOnHand: ",str(actual.QuantityOnHand))
            print(f"    ->PriceLevel1: ", str(actual.PriceLevel1))
            print(f"    ->PriceLevel2: ",str(actual.PriceLevel2))
            print(f"    ->PriceLevel3: ",str(actual.PriceLevel3))
            print(f"    ->LastTotalUnitCost: ", str(actual.LastTotalUnitCost))
            print(f"    ->MarginLevel1: ", str(MarginLevel1))
            print(f"    ->MarginLevel2: ", str(MarginLevel2))
            print(f"    ->MarginLevel3: ", str(MarginLevel3))
            print(f"    ->ValorInventario: ", str(ValorInventario))
            print("     ------------------------------")

Lista.agregar(Datos(str(actual.ItemCode),str(actual.QuantityOnHand),str(actual.PriceLevel
1), str(actual.PriceLevel2),str(actual.PriceLevel3),str(actual.LastTotalUnitCost)
,str(MarginLevel1),str(MarginLevel2),str(MarginLevel3),str(ValorInventario)))
            actual = actual.siguiente
```



Código para realizar los cálculos de los márgenes