Grocery Store Checkout App by Chisom Maduka

Your friend operates a grocery store and sells the following items:

1. Beverages: chocolate drinks, coffee, tea, soy drinks, pop and soda
2. Phone accessories: carrying case, earpieces, screen guards
3. Toiletries: toilet paper, body soap, scrubs, body cream, shampoo
4. Pastry: pizza, burgers, donuts, muffins,cheesecakes
5. Cosmetics: perfumes, vanishes, nail polish, deodorants, facial scrubs

Your friend wants to be able to record each sale and automatically compute total sale for a customer at checkout and generate a receipt for the customer. As part of the requirements,

1. App should store information about the products by category in the store
2. Store automatically update inventory of each product after sale or restocking
3. Raise an alert if any product inventory falls below 5 pieces
4. Store information about the purchase cost of each product and the sale price per unit
5. Allow the store owner to enter sales per item for each customer and generate a total sales
6. For each customer sales checkout:

    a. Record sales by item and sales value
    b. Show total sales by product
    c. Show total sales by category
    d. Show total sales for each day

TASK: Use your acquired knowledge of python to implement the above requirements. Please note that this MUST be a script and not a GUI application. Use python variables, containers, user input functions, functions, conditional statements and loops as necessary.

TO BUILD AN APP THAT MANAGES INVENTORY AND SALES

1. INVENTORY MANAGEMENT

1.1. Inventory Records

1.1.1. Application Workflow for Inventory Records

1.2. Inventory Updates

1.2.1. Product Restocking
1.2.2. Sales Reduction
1.2.3. Raise an alert if any product inventory falls below 5 pieces
1.2.4. Create a new category for the items not found

2. SALES RECEIPT GENERATION

2.1. Compute sales by items and for total items
2.2. Update inventory and sales record per item

1. INVENTORY MANAGEMENT

1.1. Inventory Records

1.1.1. Application Workflow for Inventory Records

1. Define and store product categories in a tuple
2. Create a dictionary to hold products by category
3. Create a dictionary for item inventory
4. Loop through any kind of product from the dictionary and check its availability and corresponding inventory

In [1]:

```
# 1. Define and store product categories in a tuple

prodCat=('beverages','PhoneAccessories', 'toiletries','pastry', 'cosmetics')

# 2. Create a dictionary to hold products by category

prodDict={'beverages':['chocDrinks', 'coffee', 'tea', 'soyDrinks', 'pop', 'soda'],
          'PhoneAccessories':['carrynCase', 'earpiece', 'scrnGuards'],
          'toiletries':['toiletPaper', 'bodySoap', 'scrubs', 'bodyCream', 'shampoo'],
          'pastry':['pizza', 'burgers', 'donuts', 'muffins','cheesecakes'],
          'cosmetics':['perfumes', 'vanishes', 'nailPolish', 'deodorants', 'facialScrubs']
         }
```

In [2]:

```
'''Testing the code for product categories in the dictionary'''

prodDict.keys()
```

Out[2]:

```
dict_keys(['beverages', 'PhoneAccessories', 'toiletries', 'pastry', 'cosmetics'])
```

In [3]:

```
'''Testing for each item in a product category'''

prodDict['PhoneAccessories']
```

Out[3]:

```
['carrynCase', 'earpiece', 'scrnGuards']
```

In [4]:

```
'''Referencing with index position'''

list(prodDict['beverages'])[0]
```

Out[4]:

```
'chocDrinks'
```

In [5]:

```
'''call inventory for product categories from the product dictionary'''

for k, v in prodDict.items():
    print(f'the product category is {k}')
    for prods in v:
          print(f'{prods} is a member of {k}')
```

```
the product category is beverages
chocDrinks is a member of beverages
coffee is a member of beverages
tea is a member of beverages
soyDrinks is a member of beverages
pop is a member of beverages
soda is a member of beverages
the product category is PhoneAccessories
carrynCase is a member of PhoneAccessories
earpiece is a member of PhoneAccessories
scrnGuards is a member of PhoneAccessories
the product category is toiletries
toiletPaper is a member of toiletries
bodySoap is a member of toiletries
scrubs is a member of toiletries
bodyCream is a member of toiletries
shampoo is a member of toiletries
the product category is pastry
pizza is a member of pastry
burgers is a member of pastry
donuts is a member of pastry
muffins is a member of pastry
cheesecakes is a member of pastry
the product category is cosmetics
perfumes is a member of cosmetics
vanishes is a member of cosmetics
nailPolish is a member of cosmetics
deodorants is a member of cosmetics
facialScrubs is a member of cosmetics
```

In [6]:

```
# 3. Creating the product inventory dictionaries

bevDict= {'chocDrinks':111, 'coffee':52, 'tea':63, 'soyDrinks':14, 'pop':145, 'soda':46}
phoneAccDict= {'carrynCase':37, 'earpiece':78, 'scrnGuards':90}
toiDict= {'toiletPaper':100, 'bodySoap':161, 'scrubs':32, 'bodyCream':33, 'shampoo':84}
pasDict= {'pizza':25, 'burgers':76, 'donuts':17, 'muffins':98,'cheesecakes':89}
cosDict={'perfumes':70, 'vanishes':21, 'nailPolish':62, 'deodorants':43, 'facialScrubs':114}
misDict={}
```

In [7]:

```
'''To confirm product availability in the inventory'''

prodCheck = 'chocolate'

if prodCheck in prodDict['beverages']:
    currInvent = bevDict[prodCheck]
    print(currInvent)

else:
    print(f'{prodCheck}, not found in inventory')
```

```
chocolate, not found in inventory
```

In [8]:

```
'''accessing the dictionary to get the products and the items'''

for k, v in prodDict.items():
    print(k,v)
```

```
beverages ['chocDrinks', 'coffee', 'tea', 'soyDrinks', 'pop', 'soda']
PhoneAccessories ['carrynCase', 'earpiece', 'scrnGuards']
toiletries ['toiletPaper', 'bodySoap', 'scrubs', 'bodyCream', 'shampoo']
pastry ['pizza', 'burgers', 'donuts', 'muffins', 'cheesecakes']
cosmetics ['perfumes', 'vanishes', 'nailPolish', 'deodorants', 'facialScrubs']
```

In [11]:

```
# 4. Loop through any kind of product from the dictionary and check its corresponding inventory
'''Creating a function that receives input and returns output'''

def prodInvent(product):
    for k,v in prodDict.items():
        category=k
        prodList=v
        prodQty=' '
        if product in prodList:
            if category=='beverages':
                prodQty= bevDict[product]
                return prodQty
            elif category=='PhoneAccessories':
                prodQty= phoneAccDict[product]
                return prodQty
            elif category=='toiletries':
                prodQty= toiDict[product]
                return prodQty
            elif category=='pastry':
                prodQty= pasDict[product]
                return prodQty
            elif category=='cosmetics':
                prodQty= cosDict[product]
                return prodQty
    return None

'''Automatically checking for the inventory of any kind of product'''

sampleProd= input('Enter Product: ')
prodCount=prodInvent(sampleProd)
if (prodCount is None):
    print (f'{sampleProd} cannot be found in the inventory. Do you want to add it?')
else:
    print (f'The current inventory for {sampleProd} is {prodCount}')
```

```
Enter Product: drinks
drinks cannot be found in the inventory. Do you want to add it?
```

1.2. Inventory Updates

1.2.1. Product Restocking

Flow: Check if the product is in the dictionary. If found, reference the corresponding dictionary, and increment the quantity by the new quantity added

In [14]:

```
# Check if the product is in the inventory. If found, reference
# the corresponding dictionary, and increment the quantity by the new quantity added
'''Creating a function that can restock products in the inventory'''

'''Code:'''
def incUpdateInvent(prod, qty):
    for k,v in prodDict.items():

        category=k
        prodList=v

        if prod in prodList:

            if category=='beverages':
                oldQty= bevDict[prod]
                bevDict[prod]+= qty  #Incrementing the bevDict with the new quantity added
                # OR bevDict['prod']= bevDict['prod'] + qty
                print ('Successfully Updated')
                return [oldQty,bevDict[prod]]

            elif category=='PhoneAccessories':
                oldQty= phoneAccDict[prod]
                phoneAccDict[prod]+= qty  #Incrementing the phoneAccDict with the new quantity added
                # OR phoneAccDict['prod']= phoneAccDict['prod'] + qty
                print ('Successfully Updated')
                return [oldQty,bevDict[prod]]

            elif category=='toiletries':
                oldQty= toiDict[prod]
                toiDict[prod]+= qty  #Incrementing the toiDict with the new quantity added
                # OR toiDict['prod']= toiDict['prod'] + qty
                print ('Successfully Updated')
                return [oldQty,toiDict[prod]]

            elif category=='pastry':
                oldQty= pasDict[prod]
                pasDict[prod]+= qty  #Incrementing the pasDict with the new quantity added
                # OR pasDict['prod']= pasDict['prod'] + qty
                print ('Successfully Updated')
                return [oldQty,pasDict[prod]]

            elif category=='cosmetics':
                oldQty= cosDict[prod]
                cosDict[prod]+= qty  #Incrementing the cosDict with the new quantity added
                # OR cosDict['prod']= cosDict['prod'] + qty
                print ('Successfully Updated')
                return [oldQty,cosDict[prod]]
        else:
            return [None, None]


'''Testing...'''

newProd= input('Enter Product: ')
quantity= int(input('Enter Quantity: '))
oldQty, newQty= incUpdateInvent(newProd, quantity)
if (oldQty is None):
    print(f'{newProd} cannot be found in the inventory. Do you want to add as a new product?')

else:
    print(f'{newProd} has been updated from {oldQty} to {newQty}')
```

```
Enter Product: coffee
Enter Quantity: 2
Successfully Updated
coffee has been updated from 50 to 52
```

1.2.2. Sales Reduction

Flow: Check if the product is in the dictionary. If found, reference the corresponding dictionary, and decrease the quantity by the new quantity sold

In [15]:

```
# Check if the product is in the dictionary. If found, reference the corresponding dictionary,
# and decrease the quantity by the new quantity sold
'''Creating a function that computes sales reduction in the inventory'''

'''Code:'''
def decUpdateInvent(prod, qty): # Updating decreasing inventory
    for k,v in prodDict.items():

        category=k
        prodList=v

        if prod in prodList:

            if category=='beverages':
                oldQty= bevDict[prod]
                bevDict[prod]-= qty  #Decreasing the bevDict with the quantity deducted
                # OR bevDict['prod']= bevDict['prod'] - qty
                print ('Successfully Updated')
                return [oldQty,bevDict[prod]]

            elif category=='PhoneAccessories':
                oldQty= phoneAccDict[prod]
                phoneAccDict[prod]-= qty  #Decreasing the bevDict with the quantity deducted
                # OR phoneAccDict['prod']= phoneAccDict['prod'] - qty
                print ('Successfully Updated')
                return [oldQty,bevDict[prod]]

            elif category=='toiletries':
                oldQty= toiDict[prod]
                toiDict[prod]-= qty  #Decreasing the bevDict with the quantity deducted
                # OR toiDict['prod']= toiDict['prod'] - qty
                print ('Successfully Updated')
                return [oldQty,toiDict[prod]]

            elif category=='pastry':
                oldQty= pasDict[prod]
                pasDict[prod]-= qty  #Incrementing the pasDict with the new quantity added
                # OR pasDict['prod']= pasDict['prod'] - qty
                print ('Successfully Updated')
                return [oldQty,pasDict[prod]]

            elif category=='cosmetics':
                oldQty= cosDict[prod]
                cosDict[prod]-= qty  #Decreasing the bevDict with the quantity deducted
                # OR cosDict['prod']= cosDict['prod'] - qty
                print ('Successfully Updated')
                return [oldQty,cosDict[prod]]
        else:
            return [None, None]


'''Testing...'''

newProd= input('Enter Product: ')
quantity= int(input('Enter Quantity: '))
oldQty, newQty= decUpdateInvent(newProd, quantity)
if (oldQty is None):
    print(f'{newProd} cannot be found in the inventory. Please try again')

else:
    print(f'{newProd} has been updated from {oldQty} to {newQty}')
```

```
Enter Product: soyDrinks
Enter Quantity: 12
Successfully Updated
soyDrinks has been updated from 15 to 3
```

1.2.3. Raise an alert if any product inventory falls below 5 pieces

Flow: Create a function that checks product availability in the inventory. If below 5, raise an alert to restock.

In [17]:

```
# Create a function that checks product availability in the inventory.
# If below 5, raise an alert to restock.

def checkinventory(prod):
    '''this function will accepts a product name and check if it is available in the inventory
    list, if found, it will return the current quantity in the inventory.'''

    if prod in prodDict['beverages']:
        return bevDict[prod]
    if prod in prodDict['phoneAcces']:
        return phoneAccDict[prod]
    if prod in prodDict['toiletries']:
        return toiDict[prod]
    if prod in prodDict['pastry']:
        return pasDict[prod]
    if prod in prodDict['cosmetics']:
        return cosDict[prod]

'''Testing...'''

prodAlert = input('enter product:  ')

prodCount = checkinventory(prodAlert)
qty= 5
if prodCount>qty:
    print (prodCount)
else:
    print(f'{prodAlert} is running out of stock!')
```

```
enter product:  soyDrinks
soyDrinks is running out of stock!
```

1.2.4. Create a new category for the items not found

Flow: If not found, prompt the user to choose a category to add the product, and update the corresponding dictionary with the product and inventory

In [23]:

```
# If not found, prompt the user to choose a category to add the product,
# and update the corresponding dictionary with the product and inventory
''' Implementation of functionality for adding new categories'''

'''Code: First create a function that allows user to create a new product category'''
prodCat=list(prodCat)
def catCreator (catName, prodInvList):

    newDict ={}
    if catName in prodCat: #Testing that the proposed category does not exist
        print(f'{catName} already exists')
    else:
        prodCat.append(catName) # Adding new category to the category list
        prodDict[catName]= prodInvList # Assigning a product list to the new category
        invName= catName[0:3] + 'Dict' # Creating new category dictionary

        if len(prodInvList)>1:
            for item in prodInvList:
                newDict[item]= 0 # For each product in a category, initialize it with zero
        elif len(prodInvList)== 1:
            newDict[prodInvList[0]]= 0

        else:
            print('Cannot create the inventory dictionary. Check the product list')

        misDict[invName]= newDict
    return [newDict, prodDict]

newCatName= input('Enter a New Category: ')
newProdList= input('Enter Product names: ')
prodPart= newProdList.split(',')
newCAT=  catCreator (newCatName, prodPart)
```

```
Enter a New Category: kitchenDict
Enter Product names: knife,spoon,fork,pot,frying pan
```

In [24]:

```
misDict
```

Out[24]:

```
{'toyDict': {'teddy bear': 0, 'water gun': 0, 'doll': 0},
 'kitDict': {'knife': 0, 'spoon': 0, 'fork': 0, 'pot': 0, 'frying pan': 0}}
```

In [25]:

```
newCAT
```

Out[25]:

```
[{'knife': 0, 'spoon': 0, 'fork': 0, 'pot': 0, 'frying pan': 0},
 {'beverages': ['chocDrinks', 'coffee', 'tea', 'soyDrinks', 'pop', 'soda'],
  'PhoneAccessories': ['carrynCase', 'earpiece', 'scrnGuards'],
  'toiletries': ['toiletPaper', 'bodySoap', 'scrubs', 'bodyCream', 'shampoo'],
  'pastry': ['pizza', 'burgers', 'donuts', 'muffins', 'cheesecakes'],
  'cosmetics': ['perfumes',
   'vanishes',
   'nailPolish',
   'deodorants',
   'facialScrubs'],
  'toyDict': ['teddy bear', 'water gun', 'doll'],
  'kitchenDict': ['knife', 'spoon', 'fork', 'pot', 'frying pan']}]
```

In [26]:

```
newCAT[0]
```

Out[26]:

```
{'knife': 0, 'spoon': 0, 'fork': 0, 'pot': 0, 'frying pan': 0}
```

2. SALES RECEIPT GENERATION

2.1. Compute sales by items and for total items

Flow:

1. Create a dictionary for the selling price of each products in the inventory
2. Allow the store owner to enter sales per item for each customer and generate a total sales

In [27]:

```
# 1. Create a dictionary for the selling price of each products in the inventory

bevSpDict= {'chocDrinks':250, 'coffee':300, 'tea':850, 'soyDrinks':400, 'pop':280, 'soda':220}
phoneAccSpDict= {'carrynCase':2200, 'earpiece':250, 'scrnGuards':300}
toiSpDict= {'toiletPaper':300, 'bodySoap':480, 'scrubs':1200, 'bodyCream':850, 'shampoo':1350}
pasSpDict= {'pizza':4500, 'burgers':1000, 'donuts':200, 'muffins':1600,'cheesecakes':1800}
cosSpDict={'perfumes':2800, 'vanishes':650, 'nailPolish':350, 'deodorants':1750, 'facialScrubs':1200}
```

In [28]:

```
# 2. Allow the store owner to enter sales per item for each customer and generate a total sales
''' Function to calculate the sales of products in the inventory'''
''' Code:'''

def prodSales (prod, qty):

    sp= ''
    totalSales=''
    if prod in prodDict['beverages']:
        sp= bevSpDict[prod]
        totalSales=sp*qty
        #return totalSales

    if prod in prodDict['PhoneAccessories']:
        sp= phoneAccSpDict[prod]
        totalSales=sp*qty
        #return totalSales

    if prod in prodDict['toiletries']:
        sp= toiSpDict[prod]
        totalSales=sp*qty
        #return totalSales

    if prod in prodDict['pastry']:
        sp= pasSpDict[prod]
        totalSales=sp*qty
        #return totalSales

    if prod in prodDict['cosmetics']:
        sp= cosSpDict[prod]
        totalSales=sp*qty
    return [sp, totalSales]

def salesFunc():
    itemList= input('Enter the list of products: ')
    itemQty= input('Enter the quantity for each products: ')
    salesDict=dict()
    prodts=itemList.split(',')
    qtyList=itemQty.split(',')
    qtys=[]

    for qty in qtyList:
        qtys.append(float(qty))
    print(prodts)
    print(qtys)
    for p,q in zip(prodts, qtys):
        print(p,q)
        prodTSales= prodSales (p,q)
        print(prodTSales)
        unitPrice=prodTSales[0]
        tSales=prodTSales[1]
        salesDict[p]=[p, q, unitPrice, tSales]
    return salesDict
```

In [29]:

```
salesFunc()
```

```
Enter the list of products: soyDrinks,coffee,tea,perfumes,pizza,earpiece
Enter the quantity for each products: 2,6,4,5,7,9
['soyDrinks', 'coffee', 'tea', 'perfumes', 'pizza', 'earpiece']
[2.0, 6.0, 4.0, 5.0, 7.0, 9.0]
soyDrinks 2.0
[400, 800.0]
coffee 6.0
[300, 1800.0]
tea 4.0
[850, 3400.0]
perfumes 5.0
[2800, 14000.0]
pizza 7.0
[4500, 31500.0]
earpiece 9.0
[250, 2250.0]
```

Out[29]:

```
{'soyDrinks': ['soyDrinks', 2.0, 400, 800.0],
 'coffee': ['coffee', 6.0, 300, 1800.0],
 'tea': ['tea', 4.0, 850, 3400.0],
 'perfumes': ['perfumes', 5.0, 2800, 14000.0],
 'pizza': ['pizza', 7.0, 4500, 31500.0],
 'earpiece': ['earpiece', 9.0, 250, 2250.0]}
```

2.2. Update inventory and sales record per item and generate receipt

Flow: Create a table containing the sales records per item; the quantity, unit price and total sales of all items

In [31]:

```
# Create a table containing the sales records per item;
# the quantity, unit price and total sales of all items
'''First import pandas library and create a dataframe for receipt generation'''

import pandas as pd
sales= salesFunc()
salesList = list(sales.values())
df= pd.DataFrame(salesList, columns=['Items','Quantity','Selling Price','Total Sales'])
print (df)
total= sum(df['Total Sales'])
dfArr=df.values.tolist()
dfArr.append(['Total','','',total])

df= pd.DataFrame(dfArr, columns=['Items','Quantity','Selling Price','Total Sales'])
df
```

```
Enter the list of products: soyDrinks,coffee,tea,perfumes,pizza,earpiece,deodorants
Enter the quantity for each products: 3,4,4,6,6,7,8
['soyDrinks', 'coffee', 'tea', 'perfumes', 'pizza', 'earpiece', 'deodorants']
[3.0, 4.0, 4.0, 6.0, 6.0, 7.0, 8.0]
soyDrinks 3.0
[400, 1200.0]
coffee 4.0
[300, 1200.0]
tea 4.0
[850, 3400.0]
perfumes 6.0
[2800, 16800.0]
pizza 6.0
[4500, 27000.0]
earpiece 7.0
[250, 1750.0]
deodorants 8.0
[1750, 14000.0]
        Items  Quantity  Selling Price  Total Sales
0   soyDrinks       3.0            400       1200.0
1      coffee       4.0            300       1200.0
2         tea       4.0            850       3400.0
3    perfumes       6.0           2800      16800.0
4       pizza       6.0           4500      27000.0
5    earpiece       7.0            250       1750.0
6  deodorants       8.0           1750      14000.0
```

Out[31]:

	Items	Quantity	Selling Price	Total Sales
0	soyDrinks	3.0	400	1200.0
1	coffee	4.0	300	1200.0
2	tea	4.0	850	3400.0
3	perfumes	6.0	2800	16800.0
4	pizza	6.0	4500	27000.0
5	earpiece	7.0	250	1750.0
6	deodorants	8.0	1750	14000.0
7	Total			65350.0