**Task 1 pseudo code:**

**MAIN FUNCTIONS ONLY**

**inputCSVShopList():**

open shop file

for each row in shop file

create temp list; shopList

get shop A store in variable

get shop B store in variable

get shop C store in variable

if in shop A

append Y to shopList

else

append N to shopList

if in shop B

append Y to shopList

else

append N to shopList

if in shop C

append Y to shopList

else

append N to shopList

append items name, cost and shopList to current shopObject

**inputCSVHouseNames():**

open file B

define gotFirstLine set to false

define list output

for each row in file

if gotFirstLine is set to false

create list CountColumn

create counter countInRange

for in range of 0 to 14

increment countInRange

append current data field to count column

set countColumn to output

set gotFirstLine to true

return output

**inputCSVShoppingList(countHouseNamesWeeks):**

for each house name in countHouseNamesWeeks

create temp list; tempStorageItemQuantity

if dayCounter smaller than 7

increment dayCounter

else

increment week counter

set dayCounter to zero

open csv file B

next row

next row

for each column

append column to tempStorageItemQuantity

append tempStorageItemQuantity, house name, weekCounter to houseObject

**categorySort():**

create list to store split words

create dictionary to store each item category and that categories items

create list of categories with uppercase letters

create list of categories with lowercase letters

for each item

get item name store in variable.

use python split function to split item name into key words store in split words list

define bool foundCatagory

for each word in split word list

for each category name

if a word in split word list matches a category in uppercase category list or lowercase

append to that category in dictionary

set foundCatagory to true

if foundCatagory equals false

append item to noCategory list in main dictionary

for each item in list of categories with uppercase letters

append category names and contents of there lists in category dictionary to category class object

**removeBlank(aList):**

for each item in aList

if current aList item equal to “”(empty)

replace with 0

return aList

**replace():**

define temp list

for each house object

call removeBlank function store result in temp list

replace old list in houseObject with temp list

reset temp list

**inStore(item):**

define inShops String variable

get items stores set to temp list

if item is in store A

return A

if item is in store B

return B

if item is in store C

return C

**minimumStores(StoreShopChoice):**

define StoreStoreAmmountsA set to 0

define StoreStoreAmmountsB set to 0

define StoreStoreAmmountsC set to 0

define logicalChoice list

for each storeShopChoice

if current storeShopChoice equal to A

increment StoreStoreAmmountsA

elseif current storeShopChoice equal to B

increment StoreStoreAmmountsB

elseif current storeShopChoice equal to C

increment StoreStoreAmmountsC

if storeStoreAmmountA is greater than or equal to 1

append A to logicalChoice list

if storeStoreAmmountB is greater than or equal to 1

append B to logicalChoice list

if storeStoreAmmountC is greater than or equal to 1

append C to logicalChoice list

return logicalChoice

**Process():**

For each house

Get all items required store in temp list

For each item in list

If the item amount required is greater than one

Call inStore function store result in variable.

Append result of instore function to list.

Get item’s name store in variable

If the item can be found in store A

Append item to that houses store A shopping list

Append item’s amount to that houses store A item amount list

If the item can be found in store B

Append item to that houses store B shopping list

Append item’s amount to that houses store B item amount list

If the item can be found in store C

Append item to that houses store B shopping list

Append item’s amount to that houses store B item amount list

Get minimum store combinations using minimumStores function store in variable

Set minimum store combinations for that house

Clear lists for next house

**Swap(itemRequired, shopToAvoid, biggestShopItemList):**

If biggest shopping list for a equal to A

Set variable to A

If biggest shopping list for a equal to B

Set variable to B

If biggest shopping list for a equal to C

Set variable to C

For each category

For each item in category

If the old item is an item in that category

If the old item is in the no substatutes category

Return old item

For each item in a category

Store new items stores in variable

If item in category is not old item and can be bought from shop with most items

Return new item

**Substitutions(houseCount):**

Find the store in which house requires most items for and store in variable.

Find the store in which house requires least items for and store in variable.

If the size of a is the smallest

Set a as smallest shop

Else if B is the smallest

Set B as smallest shop

Else if c is the smallest

Set C as smallest shop

If the size of A is the biggest

Set A as biggest shop

Else if B is the biggest

Set B as biggest shop

Else if C is the biggest

Set C as biggest shop

If A has fewest amount of items

For all items in shop A the house requires

Get all items alternatives using swap function and store in list

For all items in new items list

Get the new items number store in variable.

If new item in shop B

Append item to shop B shopping list for house.

Append item quantity to shop B shopping list for house.

If new item in shop C

Append item to shop C shopping list for house.

Append item quantity to shop C shopping list for house.

Clear items in house shopping list A

Clear item A quantities

If B has fewest amount of items

For all items in B the house requires

Get all items alternatives using swap function and store in list

For all items in new items list

Get the new items number store in variable.

If new item in shop A

Append item to shop A shopping list for house.

Append item quantity to shop A shopping list for house.

If new item in shop C

Append item to shop C shopping list for house.

Append item quantity to shop C shopping list for house.

Clear items in house shopping list A

Clear item A quantities

If C has fewest amount of items

For all items in C the house requires

Get all items alternatives using swap function and store in list

For all items in new items list

Get the new items number store in variable.

If new item in shop A

Append item to shop A shopping list for house.

Append item quantity to shop A shopping list for house.

If new item in shop B

Append item to shop B shopping list for house.

Append item quantity to shop B shopping list for house.

Clear items in house shopping list C

Clear item C quantities

**RecalculateMinShops()**

For each house

Define MinimumStoresList

If length of items required from shop A is greater than 0

Append shop A to MinimumStoresList

If length of items required from shop B is greater than 0

Append shop B to MinimumStoresList

If length of items required from shop C is greater than 0

Append shop C to MinimumStoresList

**MatchCombinations(combination, minimumCombos):**

For each combination set

Define match

For each shop in combination set

If minimumCombos equals a shop from combination

Increment match variable

If match equals the length of current combination of shops

Return true

Return false

**commonShopCombinations(getWeek):**

define combinations

for each house

if house is in week one

get minimumCombinations store in variable minimumCombos

if the length of combinations is zero

append first houses combination

else

define matchCondition set to false

matchCondition equal to matchCombinations() parse in combinations, minimumCombos

if matchCondition equals true

iterate loop one time

if matchCondition equals false

append minimumCombo to combinations.

**mergeShopLists(week, weekCombinations):**

for all weekCombinations

for all shops in weekCombinations

append shop to listOfShops

return listOfShops

**shoppingSceduleAddShops(Weeks, listofshops):**

for each in listofshops

for each in shoppingSceduleObjects

if current shoppingSceduleObject’s week is equal too week 1

if shoppingSceduleObject has no shop to buy from

set shop from listofShops current shop

break

**shoppingSceduleShoppingSort(week):**

for each house

if that houses week equals week

for each shoppingSceduleObject

if current shoppingSceduleObject’s week is equal to variable week

if shops required for a house match shopping schedule shop then

add items required to buy to that day.

add item amounts to that day.

add other shop items required to buy to the next day.

add other shop item amounts required to buy to next day

break