Virtual vending machine:

Aims:

The aim of the program is to simulate a vending which sells different items. The user is given a certain amount of money when the program starts which allows them to purchase items from the machine. When the program starts each slot will be populated with a random number of each items so there is the possibility something can be sold out. The machine is laid out in a grid and the user uses an A-D & 1-4 system to reach each item. The cost is already assign to each item and doesn't change. While an item is in stock and the user has enough money, they can keep buying things.

Analysis of requirements:

Before the user even gets to see the program on screen several operations will need to run. Firstly the program will need to read the file containing the item, quantity and price of each item, and populate the vending machine. There needs to be a way in which each item has the correct stock and price associated with it once brought into the program.

There are two separate programs and part of my assignment, part 1 is the actual vending machine which the user gets to see and interact with. The second program is for the only the owner to use, this lets them view, edit and delete items within the machine. I could have put both code into the same program but I thought it made sense to have a separate experience for the user and owner because if this were to be adapted in the future the owner's program could be modified to assess multiple vending machines over a network rather than having to individually access each machine separately.

The two programs and connected via the same pickled file which they read from to get the item, quantity and price of each item.

User experience:

I start by importing 3 python libraries ‘time’, ’pickle’ and ‘random’. I immediately use the random module to set the amount of money the user has available to spend.

```
import time
import pickle
import random

VALID_INPUT = ["A1","A2","A3","A4",
               "B1","B2","B3","B4",
               "C1","C2","C3","C4",
               "D1","D2","D3","D4"]

money = random.range(3,8)
selection = ""

def import_content():
    file = open("inventory.dat","rb")

    global item
    global quantity
    global price

    item = pickle.load(file)
    quantity = pickle.load(file)
    price = pickle.load(file)
    file.close()
```

The program starts with the import_content function which assess the pickled file and the 3 lists it contains, it saves these into global variables ‘item’, ’quantity’ and ‘price’ so they can be assessed by all the other functions in the code.

Once the data has been imported I need to display it on screen. I tried out a few different method of printing it and this seemed the best.

Display board:

```
def display_board():

    print("You have £",money, "left\n")

    for i,x,y,z in zip(VALID_INPUT,item,quantity,price):
        print (i," ",x,"  ",y,"  ","£",z)
```

It is a for loop using the built in zip function, this function zips together multiple lists so instead of looping through and print out all of once list, the all of the next list, it print the first item of each list, then the second item of the each list and so on until complete. However, it will only print out as long as each list still has an item to print, if even one of the lists doesn’t have another item to print it won’t print the next of any. I use this to my advantage as it will only print out as for as many items and there are in the item, quantity and price lists, which means it doesn’t print out all 16 VALID_INPUTS each time.

The result the user sees looking like this:

```
You have £ 4 left

A1   Twix    7    £ 0.6
A2   Crunchie    3    £ 0.9
A3   Twirl    7    £ 0.8
A4   Kitkat    0    £ 0.75
Choose your item
```

Main:

Next is the main section of the code, from each of the functions are called. It begins by calling the import_content and display_board previously discussed.

```
import_content()
display_board()

while selection !="QUIT":
    selection = input("Choose your item ")
    selection = selection.upper()

    if selection in VALID_INPUT:
        item_id = VALID_INPUT.index(selection)

        if quantity[item_id] != 0:
            pass
            money = user_cash(money,price[item_id],quantity[item_id])
        if money > price[item_id]:
            pass
            quantity[item_id] = new_stock(money,quantity[item_id],item[item_id],

        sale(item[item_id],quantity[item_id])

        display_board()
    elif selection == "QUIT":
        break
    else:
        print("Please enter a valid selection")

save()
```

The whole thing is contained within a while loop so it continues allowing the user to purchase items until they want to Quit.

I get the user to input the item code of the product they want to buy and check if the item code is correct by checking if it is in the VALID_INPUT list. If it isn't then it will ask for another valid input, but if it is valid then it will find the index(position) of their choice in the VALID_INPUT list. For example if they choose A3 then item_id will become 2.

Because each I zipped together the lists earlier each I know each item in valid input relates to an item, price and quantity. A1 = first item, A2 = second item etc.

The 'pass' statements are there so to fix an issue I was having. When an item was out of stock it would still continue to run the next piece of code in the loop and cause it to think it was both out of stock and the user not have enough money, when they actually had enough but couldn't purchase it. The pass statement skips over bits of the loop if a condition is true.

User cash:

I run the 'user_cash' function and because item_id now relates to the position of the correct item I use this to input the correct price and quantity into the function, along with the users current money:

```
def user_cash(money,item_price,quantity):
    if money>item_price and quantity!=0:
        return money-item_price
    else:
        print("sorry, you dont have the funds")
        return money
```

The function checks if the amount of money the user has is greater than the price of the item AND that the item is still in stock, if they are both true then it will minus the cost of the item from their current balance. If either of these are false then it simply returns the same money they had before along with a message to tell them they don't have enough money.

New stock:

Next I call the new_stock function, it take the uses the item_id again to find the correct item, quantity and price, it also import the money again.

```
def new_stock(money,instock,item,item_price):
    if instock !=0 and money>item_price:
        print("You purchased a",item)
        return instock-1
    else:
        print("Sorry, we are sold out")
        return instock
```

If instock (the quantity of the item) is not out of stock AND the users money is greater than the cost of the item, then it will take 1 away from the inventory. If it is out of stock they it will return 0 and tell them it's out of stock.

Sale:

Finally the loop runs the sale function, this takes the name of the item and its current quantity so mark the date and time of which the sale was made.

This uses the time function to find the date using time.strftime and format it like so: 27/11/2014 saves it as a string, it then does the same for the current time. Next it opens up the sales file to append it, I have to convert the inventory(quantity) to a string because its currently a integer, then I can write to the file so that it show the item, stock and time of sale. This is not accessible by the user and is only for the purpose of the owner.

Save:

Once the loop has been exited because they want to quit then I run the save function which opens up the pickled file and dumps all 3 lists back into the file.

```
def save():
    file_write = open("inventory.dat", "wb")
    pickle.dump(item,file_write)
    pickle.dump(quantity,file_write)
    pickle.dump(price,file_write)
    file_write.close()
```

Owner interface:

As this program has many of the same elements as the main vending machine I was able to take the functions I had built in that code and use them without modification in this code.

Once the program has started the owner will see a list of options from which they can decide what to do, each of these options allow them to view and edit the machine.

```
ALERT:  Kitkat needs restocking

    Vending Machine

    0 - Exit
    1 - Show Stock
    2 - Add an Item
    3 - Remove Item
    4 - Restock Item
    5 - View Recent Sales

Choice:
```

Alert:

When the program first start it immediately runs the get_inventory (same as the other code) to load in the data from the pickles. Next it runs the alert function, this is a zipped list of the name of the items and the quantity of each item. It goes through and checks if any of the items inventories are equal to 0, if so it will display an alert when the owner firsts loads the program up.

```
def alert():
    for x,y in zip(item,quantity):
        if y == 0:
            print("ALERT: ",x,"needs restocking")
```

```
choice = None
while choice != "0":

    print(
    """
    Vending Machine

    0 - Exit
    1 - Show Stock
```

Then it enters the main while loop, and while the don't want to exit it will keep asking them for a new input.

The first of this options (1) simply displays the vending machine the same as before so they can see what it looks like to the user.

Add Item:

The second option however allows them to add a new item to the vending machine. This is done by asking for a new item name, price and quantity, then simply appending these to the end of each of the 3 lists. Exactly the same as before I run the save function afterwards.

Remove item:

```
def add_new_item():
    new_item = input("Enter new Item ")
    new_quantity = int(input("Enter new Quantity "))
    new_price = float(input("Enter new Price "))

    item.append(new_item)
    quantity.append(new_quantity)
    price.append(new_price)

    save()
```

```
def remove_item():
    remove_item = input("Enter the item to remove ")
    if remove_item in item:
        position = item.index(remove_item)

        item.remove(item[position])
        quantity.remove(quantity[position])
        price.remove(price[position])

    else:
        print(remove_item, "Item not found")

    save()
```

The remove function works in a similar way instead utilizing list.remove. It first ask for the name of the item which they want rid of, then run a check to make sure it's there. Next it finds the index number of that item name and finally runs list.remove on each list in that particular index to remove the correct item,quantity and price and finally it saves again.

Restock item:

To restock an it runs fairly similar to removing an item at first. It asks for an item, checks if it's valid, then finds its position. It will then print out how many are instock of that item before prompting the user to enter a new quantity number and then updating the quantity of that item.

```
def restock_item():
    restock = input("Enter the item to restock ")
    if restock in item:
        position = item.index(restock)
        print("There are currently ",quantity[position],"in stock")
    else:
        print(restock, ": Item not found")

    new_stock = int(input("Enter new stock number "))
    quantity[position] = new_stock
    print("Item restocked")

    save()
```

Recent sales:

Recent sales can be viewed inside of out of the program because they are only stored in a txt file. All

```
def sale():
    sales = open("sales.txt","r")
    for i in sales:
        print(i.rstrip())
```

the code here is doing is opening this file and printing out each line, It also using restriping to get rid of the extra new line between each item in the file when it is read.