Module: CST 1510- Programming for Data Communication and Networks

2020 September Intake

Final Coursework – Vending Machine

Student Number: M00734701

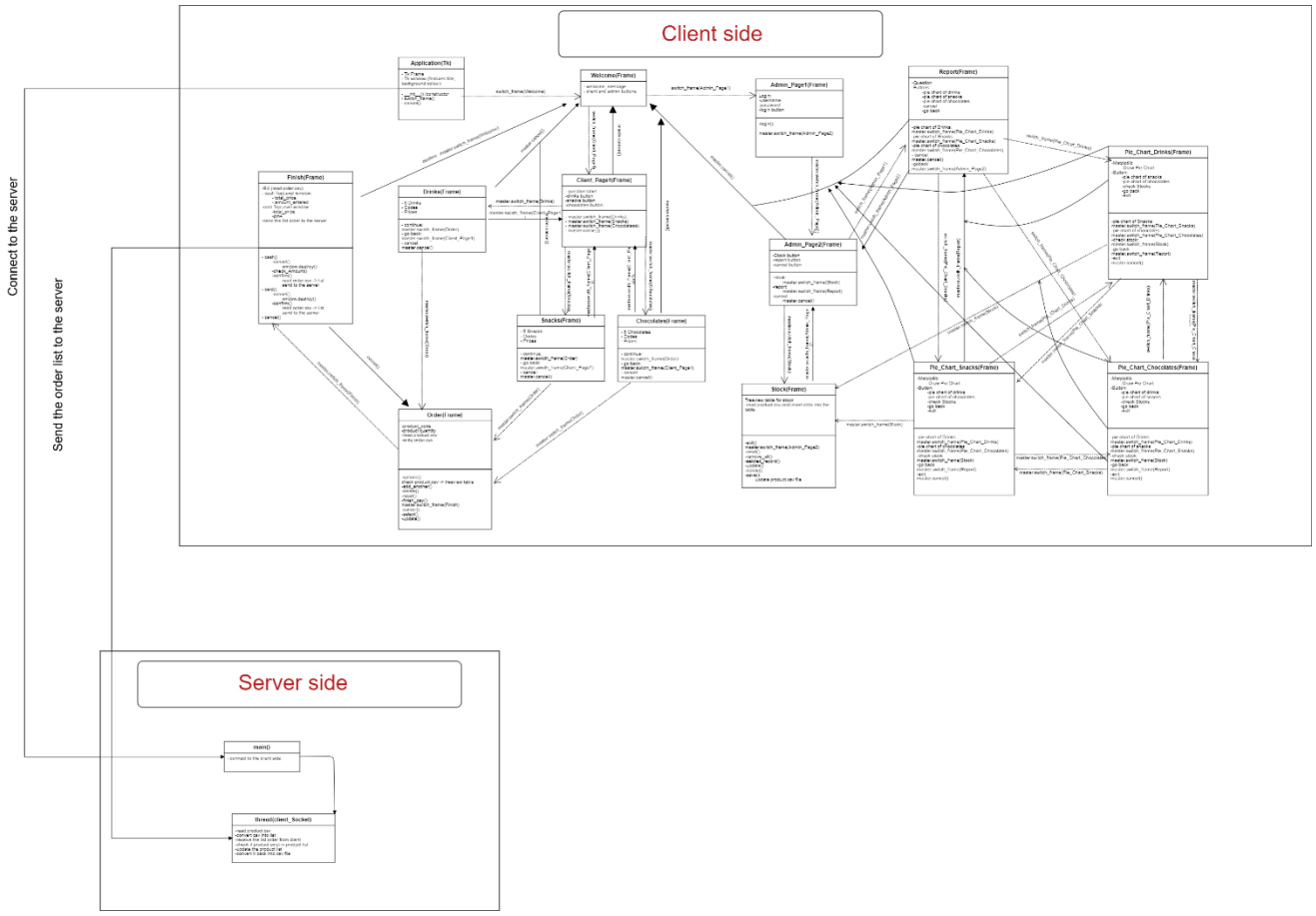
Due Date: 9th April 2021

Table of Contents

Introduction:

Using basic tools of Python programming language, and Tkinter, as standard GUI (Graphical User Interface) package, a vending machine was built by means of plots of basic classes and functions. This program allows two different options or users, the first is the client access. From this user can order any products available in the system and proceed payment. The second option is the admin, this one is a secured access where they can search, check or update the stocks. For display the orders, or stocks in a table, Treeview Widget was used and for the pie charts, Matplotlib library was used.

1. UML Diagram:



2. Implementation:

2.1. Libraries:

```
from tkinter import *
import pickle
import socket
import tkinter.messagebox
from tkinter import ttk
import pandas as pd
import csv
import numpy as np
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure

from _thread import *
```

Libraries	Description
From tkinter import *	Tkinter is a standard GUI Library for Python. ***Import **** means we will import everything from Tkinter library.
Import pickle	Used to send data over socket connection
Import socket	Used to allow the communication between the server and client
Import tkinter.messagebox	Used to display a message
From tkinter import ttk	Used to display the treeview table
Import pandas as pd	Pandas is a flexible open-source data analysis. It will be used in the program for manipulating the data and analyze them.
Import csv	CSV module is a built-in function that allows Python to store a large number of variables.
Import numpy as np	Used in the program for manipulating the data with the pandas library.
From matplotlib.backends.backend_tkagg import FigureCanvasTkAgg	To embed a matplotlib graph to python Tkinter.
From matplotlib.figure import Figure	Label for the graph
From _thread import *	Multithreading module. It will be used in the server side to avoid any types of errors.

2.2. Socket - Communication Client/Server:

The socket module provides all the required functionalities to quickly write TCP and UDP clients and servers. When we work with sockets, most applications use the concept of client/ server where there are two applications, one acting as a server and the other one as a client. Both will communicate through message passing using protocols such as TCP or UDP. **Server** represents an application waiting for connection by the client and **Client** represents an application that connects to the server.

2.2.1. Server Code:

```
# initialising the HOST and PORT
HOST ='127.0.0.1'
PORT = 3333
```

First of all, we will start by initializing the HOST and PORT, which will be the same in the Client side.

```python
def main():
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
            print('Socket successfully created')

            server_socket.bind((HOST, PORT))

            server_socket.listen(1)
            print('Waiting for connection')

            while True:
                socket_client , (host, port) = server_socket.accept()
                print(f'Received connection from {host} ({port})\n')
                print(f'Connection Established., Connected from: {host}')

                start_new_thread(thread, (socket_client,))
    except socket.error as error:
        print(f'Socket creation dropped. Error = {error} ')
        server_socket.close()
```

Main function where we will establish the connection. Secondly, we create a server_socket variable from which we will create a socket through the ***socket.socket()*** method. The type of connection used is a TCP sockets as it can be seen on the code: ***socket.SOCK_STREAM***. Sockets can be also categorized by family. There are 3 options we've learned: the one based on data, the other one which is used for working with the IPv4 protocol and the last one for working with the IPv6 protocol. In this program, I used the socket for working with the IPv4 protocol, which is the ***socket.AF_INET*** as this one allows shared manipulation of the network headers. With the line ***'server_socket.bind((HOST, PORT))***, it will allow us to connect the address with the socket. The socket must be open before establishing the connection with the address.

The method ***'server_socket.listen(1)'*** accepts as parameter the maximum number of connections from the clients. ***'server_socket.accept()'*** is a method used to enable us to accept client connections and it returns a tuple with two values, that are ***socket_client*** and client address (***'(host,port)'***). With line ***'start_new_thread(thread, (socket_client,))'*** we start a new thread to keep the connection.

Lastly for this part, in case an error occurs, the server_socket will close with line ***'server_socket.close()'***.

```python
def thread(client_socket):
    try:
        while True:
            with open('Product.csv', 'r') as product_file:
                content = csv.reader(product_file)
                list_product = list(content)

            list_order = pickle.loads(client_socket.recv(1024))


            with open('Product.csv', 'w', newline='') as updated:
                updated.write('Product_ID,Product_Name,Product_Price,Product_Quantity')
                updated.write('\n')

            with open('Product.csv', 'a', newline='') as updated_file:
                write = csv.writer(updated_file, delimiter=',')

                for product_details in list_product[1:]:
                    for order_details in list_order:
                        if order_details[0] == product_details[1]:
                            new_quantity = int(product_details[3]) - int((order_details[1]))
                            product_details[3] = new_quantity

                    write.writerow(product_details)

            client_socket.send('Product Stock updated'.encode())

    except:
        client_socket.close()
```

Thread function where we will be used to receive data from the client, send data to the client as well as to update the stock file (**'Product.csv'**).

First of all, pickle will allow to send the list. With line '***list_order = pickle.loads(client_socket.recv(1024))'***, we used the method '***client_socket.recv(1024))***' to receive data from the socket. **1024** is a method argument that indicates the maximum amount of data it can receive. '***pickle.loads()***' is a method to deserialize from a string.

With line ***'client_socket.send('Product Stock updated'.encode())'***, we are sending a message to the client saying that the Product Stock has been updated.

Lastly, we will update the product.csv file by using the order list and product list. With line '***client_socket.close()'***, the program will close the client code in case an error occurs.

Output:

```
Socket successfully created
Waiting for connection
Received connection from 127.0.0.1 (53132)

Connection Established., Connected from: 127.0.0.1
```

2.2.2. Client Code:

```
# initialising the HOST and PORT
HOST = '127.0.0.1'
PORT = 3333

# step 1 Create a client socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# step 2 Listen for a connection from a server and accepted
client_socket.connect((HOST, PORT))
```

First of all, we will start by initializing the HOST and PORT, which will be the same in the server side. Secondly, we create a client socket variable from which we will create a socket through the ***socket.socket()*** method. The type of connection used is a TCP sockets as it can be seen on the code: ***socket.SOCK_STREAM***. In this program, I used the socket for working with the IPv4 protocol, which is the ***socket.AF_INET*** as this one allows shared manipulation of the network headers.

Client_socket.connect((HOST, PORT)) is used to connect the client to the server IP address.

```
client_message = pickle.dumps(list_order)
client_socket.send(client_message)
from_server = client_socket.recv(1024)
print(from_server.decode())
```

'pickle.dumps(list_order)' is used to serialize to a string. We are going to store this into the variable ***client_message***, which will be sent to the server by the line ***client_socket.send(client_message)***. With line ***'client_socket.recv(1024)'***, the client will receive a message from the server. It will print that message in the terminal with line ***'print(from_server.decode())'***.

Output:

```
Product Stock updated
```

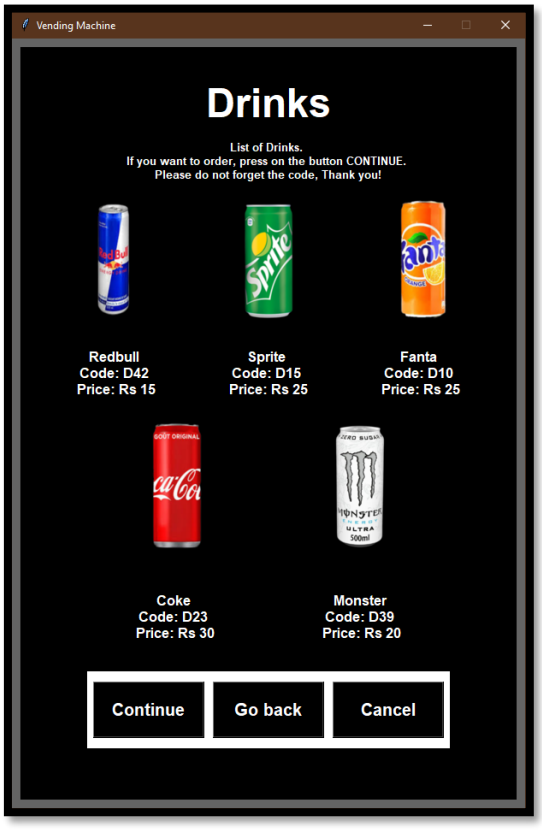
2.3. Classes:

Classes	Descriptions
Class Application(Tk)	Class initializing the main frame of windows and function to switch frame
Class Welcome(Frame)	Class to display a welcome message and buttons to chose either the user wants to continue as client or admin
Class Client_Page(Frame)	Class to choose which type of product the client wants to order
Class Drinks(Frame)	Class to display the drinks, their name, price and code
Class Snacks(Frame)	Class to display the snacks, their name, price and code
Class Chocolates(Frame)	Class to display the chocolates, their name, price and code
Class Order(Frame)	Class used to give the order
Class Finish(Frame)	Class used to do the payment. Options: Payment by cash or card
Class Admin_Page1(Frame)	Class for the admin where the user will have the login system
Class Admin_Page2(Frame)	Class to display the options either the admin wants to check the stock or have a look at the reports (pie charts)
Class Stock(Frame)	Class to display the stocks
Class Report(Frame)	Class to displays the different options for the report
Class Pie_Chart_Drinks(Frame)	Class to display the report for Drinks, using Matplotlib library
Class Pie_Chart_Snacks(Frame)	Class to display the report for Snacks, using Matplotlib library
Class Pie_Chart_Chocolates(Frame)	Class to display the report for Chocolates, using Matplotlib library

2.4. Features implemented:

2.4.1. GUI (Graphical User Interface):

'Tkinter is a great tool for developing comprehensive graphical user interfaces and for learning object-oriented programming.' (Liang, 2013) Tkinter is included with Python as a library. It gives us the ability to create Windows with widgets in them. Widget is a graphical component on the screen, such as buttons, pictures, scroll bar or text labels.



2.4.2. CSV module:

The CSV module implements classes to read and write in CSV format.

2.4.2.1. Reading in file:

```
with open('Product.csv') as product_file:
    read = csv.DictReader(product_file, delimiter=',')
    found = False
    for row in read:
        if row['Product_ID'] == code_entered.get():
            found = True
            product_name = row['Product_Name']
            productquantity = float(row['Product_Quantity'])
            product_price = row['Product_Price']
            product_price_total = float(product_price)*quantity_entered.get()
```

With line '***with open('Product.csv') as product_file'***, we are opening the csv file in read mode. This file will be read as a dictionary as it can be seen with line '***csv. DictReader'***. csv.DictReader create an object that will operate like a regular reader but maps the information in each row to a dictionary whose keys are given by the optional fieldnames parameter.

2.4.2.2. Writing in file:

```
with open('order.csv', 'w', newline='') as file:
    writer = csv.writer(file, delimiter=',')
    file.write('Product_Name,Product_Quantity, Product_Unit_Price, Product_Total_Price\n')
    for row in order.get_children():
        value = order.item(row)['values']
        writer.writerow(value)
```

With line ***'with open('order.csv','w',newline= '') as file',*** we are opening the order.csv file in writing mode. '***csv.writer'*** return a writer object responsible for converting the data into delimited strings. Csv file can be any object with a ***writer()*** method.

Here, we will extract the data from the order table which will be in the ***class Order(Frame).***with method ***order.get_children()***. And will write it into the ***order.csv*** file. The function ***'csv.writerow(row)'*** write items in a sequence separating them by comma.

2.4.2.3. Appending in file:

```
with open('Product.csv', 'a', newline='') as updated_file:
    write = csv.writer(updated_file, delimiter=',')

    for product_details in list_product[1:]:
        for order_details in list_order:
            if order_details[0] == product_details[1]:
                new_quantity = int(product_details[3]) - int((order_details[1]))
                product_details[3] = new_quantity

        write.writerow(product_details)
```

With line ***'with open('Product.csv','a',newline= '') as file',*** we are opening the order.csv file in appending mode. This part will be used to update the main stock file, which is the ***Product.csv*** file. It will be executed in the server side.

2.4.2.4. Converting the file into list:

```
with open('Product.csv', 'r') as product_file:
    content = csv.reader(product_file)
    list_product = list(content)
```

With line '***with open('Product.csv') as product_file'***, we are opening the csv file in read mode. With line ***'list_product = list(content)',*** it will convert the content into list as the function ***list()*** is applied.

2.4.3. Use of Treeview Widget for Table:

First of all, we will create the table using ***'ttk.Treeview()'*** method. With line '***columns=('p_id', 'p_name', 'p_price', 'p_quantity')'***, we are create four columns for the table. Using line ***'table.pack()'***, it will display the table in the window.

```
table = ttk.Treeview(toplevel, height=15, columns=('p_id', 'p_name', 'p_price', 'p_quantity'))
table.pack()
```

Secondly, we will assign headers to each column created previously with method ***'.heading()'***.

```
table.heading('p_id', text="Product ID")
table.heading('p_name', text="Product Name")
table.heading('p_price', text="Product Price")
table.heading('p_quantity', text="Product Quantity")
```

To display those headers into the table, we wrote the line '***table['show'] = 'headings''***.

```
table['show'] = 'headings'
```

To continue with, with function ***'.column()'*** we will define the size of each column.

```
table.column('p_id', width=80)
table.column('p_name', width=120)
table.column('p_price', width=80)
table.column('p_quantity', width=120)
```

To display the entire table, **'.pack()'** method will be used.

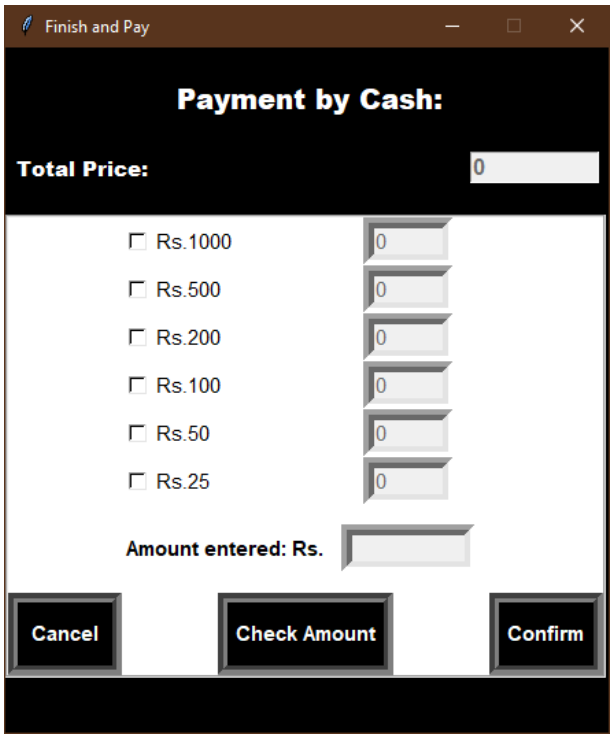
```
table.pack(fill=BOTH, expand=1)
```

2.4.4. Check buttons:

Code for check buttons: (used for the payment :type-cash)

```
rs1000 = Checkbutton(amount, text='Rs.1000', variable = value1000, onvalue=1, offvalue=0,
                     font=('Time New Roman', 11), bg='white', fg='black', command=check_rs1000)
rs1000.grid(row=0, column=1,sticky=W)
rs500 = Checkbutton(amount, text='Rs.500', variable=value500, onvalue=1, offvalue=0,
                     font=('Time New Roman', 11), bg='white', fg='black', command=check_rs500)
rs500.grid(row=1, column=1,sticky=W)
rs200 = Checkbutton(amount, text='Rs.200', variable=value200, onvalue=1, offvalue=0,
                     font=('Time New Roman', 11), bg='white', fg='black', command=check_rs200)
rs200.grid(row=2, column=1,sticky=W)
rs100 = Checkbutton(amount, text='Rs.100', variable=value100, onvalue=1, offvalue=0,
                     font=('Time New Roman', 11), bg='white', fg='black', command=check_rs100)
rs100.grid(row=3, column=1,sticky=W)
rs50 = Checkbutton(amount, text='Rs.50', variable=value50, onvalue=1, offvalue=0,
                     font=('Time New Roman', 11), bg='white', fg='black',command=check_rs50)
rs50.grid(row=4, column=1,sticky=W)
rs25 = Checkbutton(amount, text='Rs.25', variable=value25, onvalue=1, offvalue=0,
                     font=('Time New Roman', 11), bg='white', fg='black', command=check_rs25)
rs25.grid(row=5, column=1,sticky=W)
```

'Checkbutton widget is used to display a numbe of options to a user a toggle button.' The user can have the options to select one or more by clicking the corresponding button.



2.4.5. Stock Manipulation by admin:

The stock table was designed using the Treeview widget. Only the admin has access to this stock table. From the stock, the user will be able to delete one ore everything using the **delete** button and **remove all** button. The stock can be updated using first the **select** button which will fill the entry box with the data and then the user will have to click on the **update** button to apply the changements. The **reset** button is to reset the entry box when needed. To save the changes or update the csv file, it will be asked to the user to press the **save/ export** button. Lastly to quit the window, by clicking on the **exit** button, it will direct the user to the welcome window.

Delete function:

```
def delete():
    value = self.product_record.selection()[0]
    self.product_record.delete(value)
```

Remove all function:

```
def remove_all():
    for record in self.product_record.get_children():
        self.product_record.delete(record)
```

Select function:

```
def selected_record():
    self.productID_entry.delete(0, END)
    self.productName_entry.delete(0, END)
    self.productPrice_entry.delete(0, END)
    self.quantity_entry.delete(0, END)

    selected = self.product_record.focus()
    values = self.product_record.item(selected, 'values')

    self.productID_entry.insert(0, values[0])
    self.productName_entry.insert(0, values[1])
    self.productPrice_entry.insert(0, values[2])
    self.quantity_entry.insert(0, values[3])
```

Update function:

```
def update():
    selected = self.product_record.focus()
    self.product_record.item(selected, text="",
                             values=(self.productID_entry.get(), self.productName_entry.get(),
                                     self.productPrice_entry.get(), self.quantity_entry.get()))
    self.productID_entry.delete(0, END)
    self.productName_entry.delete(0, END)
    self.productPrice_entry.delete(0, END)
    self.quantity_entry.delete(0, END)
```

Reset function:

```
def reset():
    self.productID_entry.delete(0, END)
    self.productName_entry.delete(0, END)
    self.productPrice_entry.delete(0, END)
    self.quantity_entry.delete(0, END)
```

Save/export function:

```
def save():
    with open('Product.csv', 'w', newline='') as file:
        writer = csv.writer(file, delimiter=',')
        file.write('Product_ID,Product_Name,Product_Price,Product_Quantity\n')
        for row_id in self.product_record.get_children():
            row2 = self.product_record.item(row_id)['values']
            writer.writerow(row2)
        tkinter.messagebox.showinfo("Save to CSV file", "File was saved")
```

Exit function:

```
def exit():
    iexit = tkinter.messagebox.askyesno("Exit", "Confirm if you want to exit")
    if iexit > 0:
        self.destroy()
        self.master.switch_frame(Admin_Page2)
```

2.4.6. Matplotlib:

First of all, from the file ‘Product.csv’, we will create another file including only the products of the same cathegory.

```
with open('Product.csv', 'r') as file_in, open('Drink.csv', 'w') as file_out:
    reader = csv.reader(file_in)
    writer = csv.writer(file_out)
    file_out.write('Product_ID,Product_Name,Product_Price,Product_Quantity')
    file_out.write('\r\n')
    for counter, row in enumerate(reader):
        if counter < 11: continue
        if counter > 15: break
        writer.writerow(row)
```

Using pandas library, data will be analyzed:

```
data = pd.read_csv('Drink.csv')
drinks = data['Product_Name']
quantity = data['Product_Quantity']
```

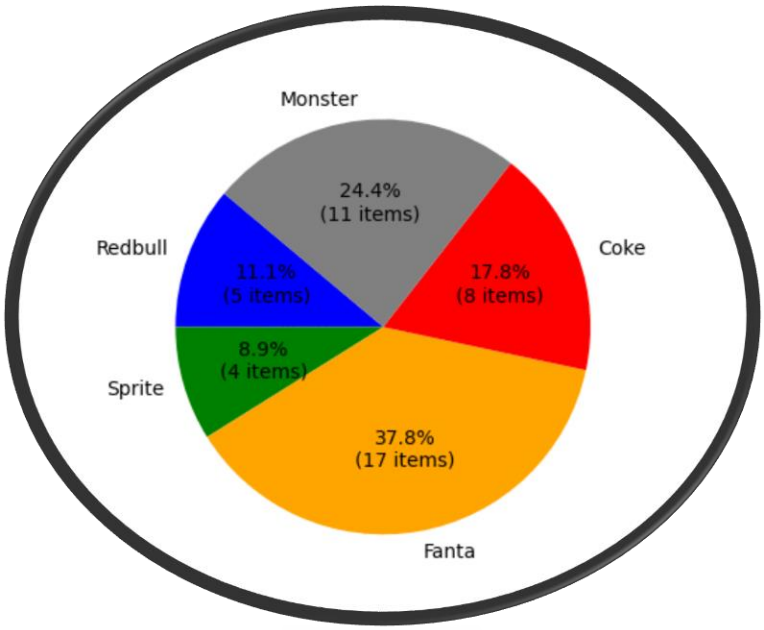
Lastly, we will draw the pie chart using **canvas** module. The method ***'.add_subplot(111)'*** is for the size of the pie chart.

```
figure = Figure()
figure.patch.set_facecolor('white')

ax = figure.add_subplot(111)

colors = ['blue', 'green', 'orange', 'red', 'grey']
ax.pie(quantity, labels=drinks, colors=colors, autopct=lambda percentage: autospt(percentage, quantity),
       startangle=140)
canvas = FigureCanvasTkAgg(figure, master = frame1)
# drawing the pie chart
canvas.draw()
# displaying the pie chart
canvas.get_tk_widget().pack()
```

Pie chart of Drinks:



3. Problems and solutions:

While doing this coursework, the problem I had to face was mostly about the communication between client and server. I didn't know how to send a csv file from the server to the client, so I watched some documentation concerning that and got the solution by converting the csv file into a list and then by using ***picle.dumps()*** method, send the file to the receiver.

The last problem I had was on how to manipulate a csv file, so to solve this problem I had to go through the slides and some tutorials.

Conclusion:

In conclusion, building this vending machine using Tkinter was simple and user friendly. Moreover, including csv file manipulation, Matplotlib library it was quite challenging.

References:

Docs.python.org. 2021. *csv — CSV File Reading and Writing — Python 3.9.4 documentation*. [online] Available at: <https://docs.python.org/3/library/csv.html> [Accessed 7 April 2021].

Liang, D., 2013. *Introduction to programming using Python*. Boston: Pearson, p.8.

Pythonbasics.org. 2021. *tkinter checkbox - Python Tutorial*. [online] Available at: <https://pythonbasics.org/tkinter-checkbox/> [Accessed 8 April 2021].

Python, R., 2021. *Python Plotting With Matplotlib (Guide) – Real Python*. [online] Realpython.com. Available at: <https://realpython.com/python-matplotlib-guide/> [Accessed 8 April 2021].

Python, R., 2021. *Socket Programming in Python (Guide) – Real Python*. [online] Realpython.com. Available at: <https://realpython.com/python-sockets/#echo-server> [Accessed 8 April 2021].